# Space vector data path.

Patent Number: EP0681236, B1
Publication date: 1995-11-08
Inventor(s): EARLE JOHN (US); GAREY KENNETH E (US); WATSON GEORGE A (US); BINDLOSS KEITH M (US)
Applicant(s):: ROCKWELL INTERNATIONAL CORP (US)
Requested Patent: JP8050575
Application Number: EP19950106843 19950505
Priority Number (s): US19940238558 19940505
IPC Classification: G06F9/302
EC Classification: G06F9/38T, G06F15/78V, G06F9/302, G06F9/318
Equivalents:

## Abstract

A space vector path for integrating SIMD scheme into a general-purpose programmable processor is disclosed. The programmable processor comprises mode means coupled to an instruction means (130, 140) for specifying for each instruction whether an operand is processed in one of vector and scalar modes, processing unit (110) coupled to the mode means for receiving the operand and, responsive to an instruction as specified by the mode means, for processing the operand in one of the vector and scalar modes, wherein the vector mode indicating to the processing unit (110) that there are a plurality of elements within the operand and the scalar mode indicating to the processing unit (110) that there is one element within the operand. 

(19)日本国特許庁（ＪＰ）　(12)公開特許公報(A)　(11)特許出願公開番号

特開平8－50575

(43)公開日　平成８年(1996)２月20日

| (51)Int.Cl.[6] | 識別記号 | 庁内整理番号 | ＦＩ | 技術表示箇所 |
|---|---|---|---|---|
| Ｇ０６Ｆ　15/80 | | | | |
| 9/38 | ３１０　Ｇ | | | |

審査請求　未請求　請求項の数30　ＯＬ　（全 34 頁）

(21)出願番号　特願平7－109642

(22)出願日　平成７年(1995)５月８日

(31)優先権主張番号　２３８５５８

(32)優先日　1994年５月５日

(33)優先権主張国　米国（US）

(71)出願人　590002448
ロックウェル・インターナショナル・コーポレイション
ROCKWELL INTERNATIONAL CORPORATION
アメリカ合衆国、90740－8250　カリフォルニア州、シール・ビーチ、シールビーチ・ブールバード、2201

(72)発明者　ケイス・エム・ビンドロス
アメリカ合衆国、92714　カリフォルニア州、アーバイン、ビーバー・ストリート、3861

(74)代理人　弁理士　深見　久郎　（外３名）

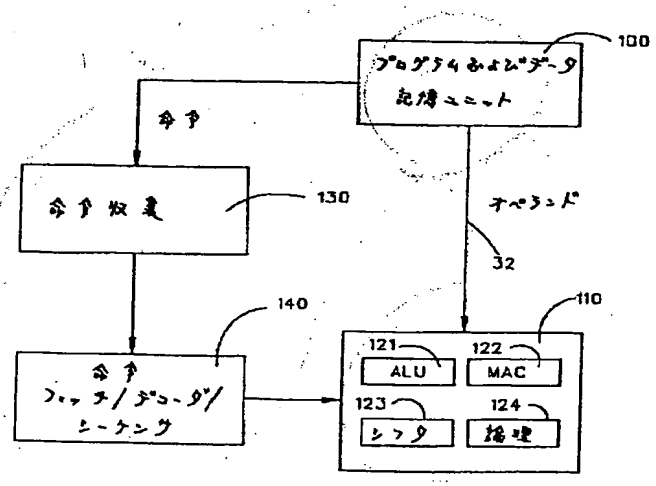
最終頁に続く

(54)【発明の名称】　プログラマブルプロセッサ、前記プログラマブルプロセッサを用いてデジタル信号処理を行なうための方法およびその改良

(57)【要約】

【目的】　汎用プログラマブルプロセッサにＳＩＭＤスキームを統合する空間ベクトルデータ経路を提供する。

【構成】　プログラマブルプロセッサは、命令手段に結合され、オペランドがベクトルおよびスカラモードの１つ処理されるか否かを各命令に対して特定するためのモード手段と、モード手段に結合され、オペランドを受取り、モード手段に特定されるような命令に応答してベクトルおよびスカラモードの１つでオペランドを処理するための処理ユニット１１０とを含み、ベクトルモードはオペランド内に複数の要素があることを処理ユニット１１０に示し、スカラモードはオペランド内に１つの要素をあることを処理ユニット１１０に示す。



【特許請求の範囲】

【請求項１】　少なくとも１つのオペランドの複数データ経路処理のためのプログラマブルプロセッサであって、各オペランドは少なくとも１つの要素を含み、前記プロセッサは命令フェッチ／デコード／シーケンス手段によって判断される予め定められるシーケンスで命令を実行し、前記プログラマブルプロセッサは、

ａ）　前記命令手段に結合され、前記少なくとも１つのオペランドがベクトルおよびスカラモードのうちの１つで処理されるかどうかを各命令に対して特定するためのモード手段と、

ｂ）　前記モード手段に結合される処理ユニットとを含み、前記処理ユニットは、前記少なくとも１つのオペランドを受取り、前記モード手段によって特定される前記命令に応答して前記ベクトルおよびスカラモードのうちの１つにある前記少なくとも１つのオペランドを処理し、前記ベクトルモードは複数の要素が前記オペランド内にあることを前記処理ユニットに示し、前記スカラモードは１つの要素が前記オペランド内にあることを前記処理ユニットに示す、プログラマブルプロセッサ。

【請求項２】　前記処理ユニットは、

ａ）　前記モード手段からの命令に応答し、前記少なくとも１つのオペランドにある各それぞれの要素を同時並列処理して、前記ベクトルモードの各それぞれの要素に対する独立した結果を得るための第１のベクトル手段と、

ｂ）　前記モード手段からの前記命令に応答し、前記少なくとも１つのオペランドにある第１の要素を、前記ベクトルモードの前記オペランドにある少なくともひとつの第２の要素との選択的組合せで処理するための第２のベクトル手段と、

ｃ）　前記モード手段からの前記命令に応答し、前記オペランドの各それぞれの部分を処理してそれぞれの部分的結果を得、各それぞれの部分的結果を併せて前記スカラモードでスカラ結果を引出すためのスカラ手段とを含む、請求項１に記載のプログラマブルプロセッサ。

【請求項３】　前記第１のベクトル手段およびスカラ手段は、

ａ）　複数の乗算累算器と、

ｂ）　複数のシフタと、

ｃ）　複数の算術ユニットと、

ｄ）　論理ユニットとのうちの少なくとも１つを備え、各々は、ベクトルオペランド内の少なくともひとつのそれぞれの要素と、スカラオペランドのそれぞれの部分とのうちの１つを処理する、請求項２に記載のプログラマブルプロセッサ。

【請求項４】　前記スカラ手段は条件付移動を行ない、前記第２のベクトル手段と前記スカラ手段とは前記第２のベクトルモードにある前記オペランド内の前記第１および第２の要素の前記選択的組合せに基づいて条件付き分岐を行なう、請求項３に記載のプログラマブルプロセッサ。

【請求項５】　前記処理ユニットは、

ａ）　前記ベクトルおよびスカラモードのうちの１つで動作する複数の加算器を備え、前記複数の加算器の各加算器は前記モード手段によって特定されるベクトルオペランドからの要素を受取りそれを個別に処理し、前記複数の加算器は前記モード手段によって特定されるスカラオペランドを受取ってそれをともに処理し、前記処理ユニットはさらに、

ｂ）　前記複数の加算器に結合され、前記複数の加算器が１つの加算器として前記スカラオペランドを処理するよう、キャリーステータスを前記スカラもどで前記複数の加算器の各々の間で送るための加算器制御手段を備える、請求項１に記載のプログラマブルプロセッサ。

【請求項６】　前記複数の加算器は１つの加算器として前記スカラオペランドを処理するよう、前記加算器制御手段は前記スカラモードで前記複数の加算器の各々の間でオーバフローステータスをさらに送る、請求項５に記載のプログラマブルプロセッサ。

【請求項７】　前記処理ユニットは、

ａ）　前記ベクトルおよびスカラモードのうちの１つで動作する複数の乗算累算器（ＭＡＣ）を備え、各ＭＡＣは前記モード手段によって特定されるようなベクトルオペランド内の要素を受取ってそれを別個に同時並列処理し、前記複数のＭＡＣは前記モード手段によって特定されるスカラオペランドを受取りそれをともに処理し、前記処理ユニットはさらに、

ｂ）　前記ＭＡＣに結合され、前記モード手段に応答し、前記複数のＭＡＣを、ベクトルモードでは互いに独立して動作させ、スカラモードではともに動作させるためのＭＡＣ制御手段を備える、請求項１に記載のプログラマブルプロセッサ。

【請求項８】　前記処理ユニットは、

ａ）　前記ベクトルおよびスカラモードのうちの１つで動作する複数の論理ユニットを備え、前記複数の論理ユニットの各論理ユニットは前記モード手段によって特定されるようなベクトルオペランド内の要素を受取ってそれを別個に同時並列処理し、前記複数の論理ユニットは前記モード手段によって特定されるスカラオペランドを受取ってそれをともに処理し、前記処理ユニットはさらに、

ｂ）　前記複数の論理ユニットに結合され、前記複数の論理ユニットが１つの論理ユニットとして前記スカラオペランドを処理するよう、前記スカラモードで前記複数の論理ユニットの各々の間でゼロステータスを送るための、論理制御手段を備える、請求項１に記載のプログラマブルプロセッサ。

【請求項９】　前記処理ユニットは、

ａ）　前記スカラモードでは１つの統合化されたシフタ

として、および前記ベクトルモードでは複数のシフタとして、選択的に動作するための複数のシフタを備え、前記複数のシフタの各々は、演算の第１のモードにある前記モード手段に応答し、特定されるベクトルオペランドからの要素を受取ってそれを別個に同時並列処理し、前記複数のシフタは、第２のモードの演算にある前記モード手段に応答し、スカラオペランドを受取ってそれをともに処理し、前記処理ユニットはさらに、

ｂ）　前記シフタに結合され、前記複数のシフタが前記スカラオペランドを処理するよう、シフトされたオペランドビットを前記スカラモードで前記複数のシフタの各々の間で送るためのシフタ制御手段を含み、前記シフタ制御手段は、前記ベクトルモードでは、前記複数のシフタの各々からシフトされたオペランドビットを送ることを不能化する、請求項１に記載のプログラマブルプロセッサ。

【請求項１０】　スカラおよびベクトルモードの１つでオペランドの条件を評価するための比較手段をさらに含む、請求項１に記載のプログラマブルプロセッサ。

【請求項１１】　前記比較手段は命令実行のシーケンスを修飾するために各オペランドの条件を評価する、請求項１０に記載のプログラマブルプロセッサ。

【請求項１２】　第１のオペランドは前記比較手段に基づいて第１の記憶位置から第２の記憶位置へ条件付きで移動され、前記比較手段は、第１のオペランド内の対応する要素が移動されるかどうかを判断するために第２および第３のオペランド内の対応する要素を各々が比較する複数のサブ比較器を含む、請求項１０に記載のプログラマブルプロセッサ。

【請求項１３】　前記モード手段は、各々の命令が命令単位でベクトルおよびスカラモードのうちの１つを特定するよう、各命令内にフィールドとして含まれる、請求項１に記載のプログラマブルプロセッサ。

【請求項１４】　前記モード手段は各命令内にビットフィールドとして含まれる、請求項１３に記載のプログラマブルプロセッサ。

【請求項１５】　各オペランド内に少なくとも１つの要素を有するオペランドをストアするためのデータメモリと、実行のための命令をストアするための命令メモリと、命令手段と、複数の算術論理ユニット（ＡＬＵ）とに結合される汎用コンピュータにおける、複数データデジタル信号処理を行なうための構成であって、

ａ）　前記命令メモリと前記命令手段とに結合され、オペランドが前記処理ユニットによってベクトルモードおよびスカラモードのうちの１つとして処理されるかどうかを各命令において特定するためのモード手段と、

ｂ）　前記モード手段に応答し、前記ＡＬＵを、スカラオペランドの場合は第１のモードで１つのユニットとしてともに動作させ、ベクトルオペランドの場合には各ユニットが第２のモードにある状態で個々の演算ユニットとて独立して動作させることを選択的に行なうためのＡＬＵ制御手段と、

ｃ）　前記ＡＬＵ制御手段と前記ＡＬＵとに結合され、スカラオペランドの場合には前記ＡＬＵの各々の間でキャリー条件を選択的に送り、ベクトルオペランドの場合は前記ＡＬＵの各々のための前記キャリー条件を無視するための、キャリー条件手段とを備える、各オペランド内に少なくとも１つの要素を有するオペランドをストアするためのデータメモリと、実行のための命令をストアするための命令メモリと、命令手段と、複数の演算論理ユニット（ＡＬＵ）とに結合される汎用コンピュータにおける、複数データデジタル信号処理を行なうための構成。

【請求項１６】　各々がその中に少なくとも１つの要素を有するオペランドをストアするためのデータメモリと、実行のための命令をストアするための命令メモリと、命令手段と、第１の乗算累算器（ＭＡＣ）とに結合される汎用コンピュータにおける、複数データデジタル信号処理を行なうための構成であって、

ａ）　前記命令メモリと前記命令手段とに結合され、オペランドが前記処理ユニットによってベクトルモードおよびスカラモードのうちの１つとして処理されるかどうかを各命令において特定するためのモード手段と、

ｂ）　複数のＭＡＣと、

ｃ）　前記第１および複数のＭＡＣの各々に結合され、前記モード手段に応答し、前記第１および複数のＭＡＣの各々を、ベクトルモードでは互いに独立して動作させ、スカラモードではともに動作させることを選択的に行なうためのＭＡＣ制御手段とを備える、複数データデジタル信号処理を行なうための構成。

【請求項１７】　ＡＬＵによるオペランドの処理は、

ａ）　前記オペランド内の各独立した要素に結合される各条件コードの組と、

ｂ）　前記オペランド内の組合せにおける選択的組合せの、条件コードの複数の組と、

ｃ）　前記スカラオペランドのための条件コードの前記１つの組とのうちの１つに基づいて修飾される、請求項１５に記載の構成。

【請求項１８】　前記命令の実行のシーケンスは、

ａ）　前記オペランド内の各独立した要素に関連する各条件コードの組と、

ｂ）　選択的組合せにある条件コードの複数の組とのうちの１つによって、第１の命令から第２の命令に修飾される、請求項１５に記載の構成。

【請求項１９】　オペランドは、

ａ）　前記オペランド内の各々の独立した要素と関連する各条件コードの組と、

ｂ）　選択的組合せにある条件コードの複数の組と、

ｃ）　前記スカラオペランドのための条件コードの前記１つの組とのうちの１つに基づいて、第１の記憶位置か

ら第２の記憶位置へ選択的に移動される、請求項１８に記載の信号プロセッサ。

【請求項２０】　ａ）　前記スカラモードでは１つの統合されたシフタとして、および前記ベクトルモードでは複数のシフタとして、選択的に動作するための複数のシフタをさらに含み、前記複数のシフタの各々は、演算の第１のモードにある前記モード手段に応答し、特定されるようなベクトルオペランドからの要素を受取ってそれを独立して処理し、前記複数のシフタは、第２のモード演算にある前記モード手段に応答し、スカラオペランドを受取ってそれをともに処理し、さらに、

ｂ）　前記シフタに結合され、前記複数のシフタが前記スカラオペランドを処理するよう、シフトされたオペランドビットを前記スカラモードで前記複数のシフタの各々の間で送るためのシフタ制御手段を含み、前記シフタ制御手段は前記ベクトルモードでは前記複数のシフタの各々からシフトされたオペランドビットを送ることを不能化する、請求項１５に記載の構成。

【請求項２１】　汎用コンピュータを用いる複数データ経路計算のためのプログラマブルプロセッサであって、前記汎用コンピュータは、オペランドをストアするためのデータメモリと、データメモリからオペランドを転送するためのメモリアクセスバスと、実行のための命令をストアするための命令メモリと、前記命令メモリに結合され前記命令のフェッチ、デコードおよび順序付けのための命令手段とを含み、前記プログラマブルプロセッサは、

ａ）　前記命令手段に結合され、データメモリからのオペランドが単一データ経路モードおよび複数データ経路モードのうちの１つで処理されるかどうかを各命令において特定するためのモード手段を含み、

ｂ）　各データ経路は、

演算ユニットと、

乗算累算器（ＭＡＣ）とを含み、前記プログラマブルプロセッサはさらに、

ｃ）　前記モード手段に応答し、各データ経路にある前記算術ユニットを、スカラオペランドの場合は１つのモードで１つのユニットとしてともに動作させ、ベクトルオペランドの場合は各ユニットが別のモードにある状態で個々の算術ユニットとして独立して動作させることを選択的に行なうための演算制御手段と、

ｄ）　前記算術制御手段と各経路にある前記算術ユニットとに結合され、スカラオペランドの場合には前記算術ユニットの各々の間でキャリー条件を選択的に送り、ベクトルオペランドの場合には各算術ユニットに対応する前記キャリー条件を不能化するためのキャリー条件手段と、

ｅ）　各ＭＡＣに結合され、前記モード手段に応答し、ベクトルモードでは各ＭＡＣを互いから独立させて動作させ、スカラモードではともに動作させることを選択的に行なうためのＭＡＣ制御手段とを含む、プログラマブルプロセッサ。

【請求項２２】　ａ）　前記スカラモードでは１つの統合されたシフタとして、および前記ベクトルモードでは複数のシフタとして、選択的に動作するための複数のシフタをさらに含み、前記複数のシフタの各々は、演算の第１のモードにある前記モード手段に応答し、特定されるベクトルオペランドからの要素を受取ってそれを独立して処理し、前記複数のシフタは、第２のモード演算にある前記モード手段に応答し、スカラオペランドを受取ってそれをともに処理し、さらに、

ｂ）　前記シフタに結合され、前記複数のシフタが前記スカラオペランドを処理するよう、シフトされたオペランドビットを前記スカラモードで前記複数のシフタの各々の間で送るためのシフタ制御手段を含み、前記シフタ制御手段は前記ベクトルモードでは前記複数のシフタの各々からシフトされたオペランドビットを送ることを不能化する、請求項２１に記載のプログラマブルプロセッサ。

【請求項２３】　プログラマブルプロセッサを用いて複数データ経路を介してデジタル信号処理を行なう方法であって、前記プログラマブルプロセッサは、少なくとも１つの要素を各々が有する少なくとも１つのオペランドで動作し、前記プログラマブルプロセッサは複数のサブ処理ユニットを有し、前記方法は、

ａ）　前記プログラマブルプロセッサによって実行されるべき命令の予め定められるシーケンスの中から命令を供給するステップと、

ｂ）　前記命令が、前記プログラマブルプロセッサによる少なくとも１つのオペランド上での処理のスカラおよびベクトルモードのうちの１つを生じさせるステップとを含み、前記スカラモードは前記少なくとも１つのオペランド内に１つの要素があることを前記プログラマブルプロセッサに示し、前記ベクトルモードは前記少なくとも１つのオペランド内に複数のサブ要素があることを前記プログラマブルプロセッサに示し、前記方法はさらに、

ｃ）　スカラモードの場合には、前記プログラマブルプロセッサの各サブ処理ユニットは、前記命令に応答し、前記オペランドのそれぞれの部分を受取って処理して部分的および中間結果を発生するステップと、

ｄ）　各サブ処理ユニットは、その中間結果を前記複数のサブ処理ユニットの間で送り、その部分的結果を他のサブ処理ユニットと併せて前記オペランドのための最終結果を発生するステップと、

ｅ）　第１の条件コードを発生して前記最終結果に対応するステップと、

ｆ）　ベクトルモードの場合には、前記プログラマブルプロセッサの各サブ処理ユニットは、前記命令に応答し、前記オペランド内の前記複数のサブ要素からそれぞ

れのサブ要素を受取ってそれを処理して、各中間結果は不能化されかつ各部分的結果はその対応する要素のための最終結果を表わす状態で部分的および中間結果を発生するステップと、

ｇ）　複数の第２の条件コードを、その各々が独立した結果に対応する状態で発生するステップとを含む、デジタル信号処理方法。

【請求項２４】　汎用コンピュータを介する複数データ経路計算のためのプログラマブルプロセッサであって、前記汎用コンピュータは、オペランドをストアするためのデータメモリと、プログラム命令をストアするための命令メモリと、命令手段とを含み、前記プログラマブルプロセッサは、

前記命令手段に結合され、データメモリからのオペランドがベクトルおよびスカラモードのうちの１つとして処理されるかどうかを特定するためのモード手段を含み、ベクトルモードは各オペランド内の複数の要素を判断し、スカラモードはオペランド内の１つの要素を判断し、前記プログラマブルプロセッサはさらに、

モード手段とデータメモリとに結合される複数の処理ユニットを含み、各処理ユニットはオペランドのそれぞれの要素を受取りそれを処理して、部分的結果および伝搬情報を得るために処理し、前記プログラマブルプロセッサはさらに、

前記ベクトルモードで動作し、前記処理ユニットに結合され、各部分的結果を各要素の処理のその最終結果として送り、伝搬情報を無視するためのベクトル手段と、

前記スカラモードで動作し、前記処理ユニットに結合され、各部分的結果と伝搬情報とを併せて各オペランドの処理のその最終結果を得るためのスカラ手段とを含む、プログラマブルプロセッサ。

【請求項２５】　各処理ユニットは処理条件を保存するための条件コードの組を備え、条件コードの前記組は、

ａ）　個々に第１のベクトルモードにある各組と、

ｂ）　第２のベクトルモードにある別の組との選択的組合せにある各組と、

ｃ）　前記スカラモードで組合されるスカラオペランドのすべての組とのうちの１つによって、プログラマブルプロセッサの処理を修飾する、請求項２４に記載のプロセッサ。

【請求項２６】　各処理ユニットは、

ａ）　算術ユニットと、

ｂ）　乗算累算器と、

ｃ）　論理オペレータと、

ｄ）　バレルシフタとのうちの少なくとも１つを備える、請求項２５に記載のプロセッサ。

【請求項２７】　前記モード手段は前記少なくとも１つのオペランドにあるビットフィールドによって特定される、請求項１に記載のプログラマブルプロセッサ。

【請求項２８】　前記モード手段は、前記少なくとも１つのオペランドが選択される方法によって特定される、請求項１に記載のプログラマブルプロセッサ。

【請求項２９】　前記モード手段は、前記少なくとも１つのオペランドのアドレスをさらに特定するメモリ位置にあるビットフィールドにおいて特定される、請求項２８に記載のプログラマブルプロセッサ。

【請求項３０】　前記モード手段に応答し、第２のオペランドがスカラモードにある状態で、第１のオペランドの各それぞれの要素をベクトルモードで処理するための第３のベクトル手段をさらに含む、請求項２に記載のプログラマブルプロセッサ。

【発明の詳細な説明】

【０００１】

【発明の分野】この発明は信号プロセッサに関し、より特定的には空間並列処理能力を備えたデジタル信号プロセッサに関する。

【０００２】

【発明の背景】近年、コンピュータ技術において、単一命令多重データ（ＳＩＭＤ）などの並列処理方式を備えるコンピュータは徐々に認識される割合を獲得してきている。ＳＩＭＤコンピュータは概念的には図１（ａ）で示すことができるものであり、ここでは複数の処理要素（ＰＥ）が１つのメインシーケンサによって監視されている。すべてのＰＥはメインシーケンサから通信される同じ命令を受取るが、別々のデータストリームからの異なったデータの組に対して動作する。図１（ｂ）に示すように、各ＰＥはそれ自身の局所メモリを備える中央処理装置（ＣＰＵ）として機能する。したがって、ＳＩＭＤコンピュータは各ＰＥのＣＰＵとともに複数の同期された算述論理ユニットを用いることによって、空間的並列性を達成することができる。一旦データが各ＰＥ内に存在するようになれば、個々のＰＥがそのデータを扱うことは比較的容易なことであるとはいえ、相互接続（図示せず）を介してすべてのＰＥ間で分配および通信を行なうことは、極めて複雑な仕事である。よって、ＳＩＭＤマシンは、通常専用とすることを念頭において設計されており、プログラミングやベクトル化の難しさのために、これらのマシンは汎用の用途には望ましくないものとなっている。

【０００３】一方で、ＳＰＡＲＣ（登録商標）、ＰｏｗｅｒＰＣ（登録商標）、および６８０００ベースのマシンなど現在の汎用計算機は、典型的には高性能グラフィック処理に際してはそれらの持つ３２ビットメモリ空間をフルに利用してはいない。たとえば、これらのマシンのバスが３２ビットの幅であるのに、映像および画像情報についてはデータは未だに１６ビット幅または８ビットピクセルで処理されるように制限されている。しかしながら、これらの汎用マシンは高級言語のソフトウェア環境におけるプログラミングの便利さのため魅力的である。したがって、デジタル信号処理に応用されるような

ＳＩＭＤのスピードの利点と、汎用ＣＰＵにおけるプログラミングの便利さとの間で、バランスをとることが望ましい。そうすれば、低性能なＳＩＭＤマシンの実現例であっても、汎用マシンに組込まれたならば、あたかも複数のスカラＣＰＵが並列に働いているかのように総合的なスループットが激しく向上するであろう。しかしながら、汎用マシンにＳＩＭＤが組込まれた場合、高められたスループットは、伝統的なＳＩＭＤマシンに見られる複数ユニットのスカラＣＰＵと典型的にはかかわりのある、シリコンの使用と引替えにもたらされるものではない。

【０００４】したがって、コード強調応用およびスピード強調計算のためのＳＩＭＤ能力を備える汎用プロセッサを有することが望ましいだろう。

【０００５】本発明の目的は、ＳＩＭＤ方式を汎用ＣＰＵアーキテクチャに組入れてスループットを高めることである。

【０００６】実質的にシリコンの使用を招くことなくスループットを高めることも本発明の目的である。

【０００７】この発明のさらなる目的は、同じ命令実行速度で各命令において処理されるデータ要素の数に比例してスループットを増大させることである。

【０００８】

【発明の概要】ＳＩＭＤ方式を汎用プログラマブルプロセッサに組入れるための空間ベクトルデータ経路が開示される。プログラマブルプロセッサは、命令手段に結合され、オペランドがベクトルおよびスカラモードの１つにおいて処理されるかどうかを各命令について特定するためのモード手段と、モード手段に結合され、オペランドを受取り、モード手段によって特定された命令に応答して、オペランドをベクトルおよびスカラモードのうち１つにおいて処理するための処理ユニットとを備え、ベクトルモードは処理ユニットに、オペランド内に複数個の要素があることを示し、スカラモードは処理ユニットに、オペランド内に１つの要素があることを示す。

【０００９】本発明はまた、汎用コンピュータを用いて複数のデータ経路を介しデジタル信号処理を行なう方法をも開示するものであって、汎用コンピュータは、各オペランドが少なくとも１つの要素を有する状態で複数個のオペランドをストアするためのデータメモリと、複数個のサブ処理ユニットを有する処理ユニットとを含む。この方法は以下のステップを含む。ａ）処理ユニットによって実行されるべき命令の予め定められたシーケンスの中から命令を提供する。ｂ）命令はオペランドに対する処理ユニットによる処理についてスカラモードおよびベクトルモードのうち１つを特定する。スカラモードはオペランド内に１つの要素があることを処理ユニットに示し、ベクトルモードは複数個のサブ要素がオペランド内にあることを前記処理ユニット示す。ｃ）スカラモードの場合、処理ユニットにおける各サブ処理ユニットは命令に応答して処理すべきオペランドのそれぞれの部分を受取り、部分的中間結果を発生する。ｄ）各サブ処理ユニットは複数のサブ処理ユニット間にその中間結果を送り、その部分的結果を他のサブ処理ユニットと合せて、オペランドのための最終的な結果を発生する。ｅ）最終的結果に対応するように第１の条件コードを発生する。ｆ）ベクトルモードの場合、処理ユニットにおける各サブ処理ユニットは命令に応答してオペランド内の複数個のサブ要素からそれぞれのサブ要素を受取りかつそれを処理して、部分的中間結果を発生し、各中間結果は不能化され、各部分的結果はその対応する要素のための最終的結果を表わす。ｇ）複数個の第２の条件コードを発生する。ここで第２の条件コードの各々は独立した結果に対応する。

【００１０】

【発明の詳しい説明】

一般的な実現例の考察

ＳＩＭＤ方式を汎用マシンに組込む場合、望ましくは考慮されるべきである問題がいくつかある。

【００１１】１）スカラまたはベクトルの動作の選択は、好ましくは、ある期間ベクトルモードに切換わるのではなく、命令単位で行なわれるべきである。なぜなら、いくつかのアルゴリズムはベクトルサイズが大きいと容易にベクトル化されないからである。また、ベクトル演算が選択される場合、ベクトルの次元を特定しなければならない。

【００１２】現在、本発明に従い、スカラ／ベクトルについての情報はＳＩＭＤ能力を有する各命令内のデータタイプ修飾子フィールドによって特定される。たとえば、命令はワードまたはハーフワード対演算を特定することのできる１ビット「経路」修飾子フィールドを特徴としていてもよい。さらに、より大きいベクトル次元、たとえば４、８などを選択するために、このフィールドは好ましくはストリーマコンテキストレジスタ内のデータタイプ変換フィールドと組合せられるべきである。ストリーマの完全な説明は、「ＲＩＳＣデジタル信号プロセッサのためのストリーマ（STREAMER FOR DIGITAL SIGNAL PROCESSOR）」と題され、その開示がここに引用によって援用される、１９９２年７月２３日に提出された関連の米国特許出願連続番号第９１７，８７２号に開示されている。

【００１３】２）マシンは、ベクトル結果に基づく条件付実行に備えるものでなければならない。ＳＩＭＤ演算の結果を、それがちょうど多重スカラ演算を用いて行なわれたかのようにテストできることが重要である。この理由により、ステータスレジスタ内の条件コードフラグは、データ経路の１セグメントごとに１組が存在するように二重にされることが好ましい。たとえば、ベクトル次元が４であれば４組の条件コードが必要であろう。

【００１４】また、条件付命令は、条件コードのどの組

を使用するかを特定することを必要とする。たとえば「１つでもけた上げフラグがセットされていれば」または「すべてのけた上げフラグがセットされていれば」などの条件の組合せをテストすることができれば有用である。

【００１５】３）ＳＩＭＤ方式は可能な限り多くの演算に応用可能であるべきである。これから述べる本発明の好ましい実施例は、１６ビット乗算器および３２ビット入力データなどの現在の実現例におけるマシンを示しているが、本発明に従い他の変形が容易に構成され得るということは当業者には認識されるであろう。

【００１６】次の演算は、空間ベクトル（ＳＶ）技術の性能を高めることができる、可能な演算（図２８～３６で列挙）の例である。

【００１７】ＡＢＳ，ＮＥＧ，ＮＯＴ，ＰＡＲ，ＲＥＶ，ＡＤＤ，ＳＵＢ，ＳＵＢＲ，ＡＳＣ，ＭＩＮ，ＭＡＸ，Ｔｃｏｎｄ
ＳＢＩＴ，ＣＢＩＴ，ＩＢＩＴ，ＴＢＺ，ＴＢＮＺ
ＡＣＣ，ＡＣＣＮ，ＭＵＬ，ＭＡＣ，ＭＡＣＮ，ＵＭＵＬ，ＵＭＡＣ
ＡＮＤ，ＡＮＤＮ，ＯＲ，ＸＯＲ，ＸＯＲＣ
ＳＨＲ，ＳＨＬ，ＳＨＲＡ，ＳＨＲＣ，ＲＯＲ，ＲＯＬ
Ｂｃｏｎｄ
ＬＯＡＤ，ＳＴＯＲＥ，ＭＯＶＥ，Ｍｃｏｎｄ
ここでｃｏｎｄは、ＣＣ，ＣＳ，ＶＣ，ＶＳ，ＺＣおよびＺＳであってもよい。

【００１８】４）メモリデータ帯域幅はＳＩＭＤデータ経路の性能に適合可能であるべきである。

【００１９】メモリおよびバス帯域幅をハードウェアの複雑さを増大させることなく空間ベクトルデータ経路のデータ要求に適合させることが望ましい。現在実現されているマシンにおけるデュアルアクセスの３２ビットメモリを備える２つの３２ビットバスは、算述論理ユニット（ＡＬＵ）およびデュアル１６×１６乗算／累算ユニット（ＭＡＣ）によく適合している。これらはまた、４つの８×８ＭＡＣにもよく適合するだろう。

【００２０】５）実現されるいかなる付加および変形も、付加的なハードウェアの複雑さは最小限で性能を最大限にすることによって、コスト効率の良さをもたらすべきである。

【００２１】加算器／減算器は、けた上げ伝播を止め、条件コード論理を二重にすることによって、空間ベクトルモードにおいて動作させることができる。

【００２２】シフタは、ラップアラウンド論理をも再構成し、条件コード論理を二重にすることによって空間ベクトルモードにおいて動作させることができる。

【００２３】ビット論理ユニットは、条件コード論理を二重にするだけで空間ベクトルモードにおいて動作させることができる。

【００２４】空間ベクトル条件付移動動作は、条件コードフラグのベクトルを用いてマルチプレクサを制御し、ベクトルの各要素が独立的に移動させられるようにすることによって達成され得る。

【００２５】空間ベクトルの乗算は、乗算器アレイを二重にし、部分積を組合せることを必要とする。たとえば適切な組合せ論理を備える４つの１６×８乗算器は、４つの１６×８または２つの１６×１６のベクトル演算、もしくは１つの３２×１６スカラ演算を行なうのに用いることができる。空間ベクトル乗算－累算演算はまた、けた上げ伝播を止め、条件コード論理を二重にすることができる累算加算器を、ベクトル化された累算器レジスタと同様に必要とする。

【００２６】６）汎用コンピュータにおける空間ベクトルの実現に起因するプログラミングの複雑さは、最小限にされるべきである。空間ベクトル結果をスカラ結果に組合せるために命令を考え出すことができる。

【００２７】ACC Az,Ax,Ay　累算器を加算する。SA Ay,Mz　スケーリングされた累算器対をメモリにストアする。

【００２８】MAR Rz, Ax　スケーリングされた累算器対をレジスタに移動させる。７）ベクトルが物理的メモリ境界と交差するとき、ベクトルへのアクセスがそれでも可能であるべきである。たたみ込みなどのいくつかのアルゴリズムは、データアレイを介しての増分を必要とする。アレイが長さＮのベクトルとして扱われる場合、ベクトルが部分的に１つの物理的メモリ位置の中に存在し、部分的に隣接する物理的メモリ位置の中に存在するということがあり得る。そのような空間ベクトル演算に対する性能を維持するには、物理的境界と交差するデータアクセスに対処するようにメモリを設計するか、または前述の米国特許出願「ＲＩＳＣデジタル信号プロセッサのためのストリーマ」に記載されたようなストリーマを用いることが好ましい。

【００２９】全体システム

図２は、本発明の空間ベクトルデータ経路を組入れてもよいプログラマブルプロセッサを一般化して表わしたものである。本発明に取入れられたコンセプトの１つは、スカラオペランドまたはアレイの要素に一度に１つずつ対処するように設計されたコンピュータを変形し、同時に１つより多くのオペランドを処理できるようにすることによって、その性能を高めることができるということである。

【００３０】図２に示されているのは、プログラムおよびデータオペランドをストアするためのプログラムおよびデータ記憶ユニット１００を有するプログラマブルプロセッサ、または広い意味でいう「コンピュータ」である。命令収集ユニット１３０が記憶ユニット１００から命令をフェッチし、これは命令フェッチ／デコード／シーケンスユニット１４０によりデコードかつ解釈され、処理ユニット１１０によって実行される。このようにし

て処理ユニット１１０は記憶ユニット１００から供給されるオペランドで命令を実行する。

【００３１】性能を高めるため、オペランドがスカラであるかベクトルであるかを特定するためのビットが各命令の中にある。また、それらがベクトルである場合、各オペランド内にいくつの要素があるかが特定される。この情報は典型的なデコードされた命令とともに処理ユニット１１０に送られるので、処理ユニット１１０はオペランドをスカラとして処理するべきがベクトルとして処理するべきかを「知る」。

【００３２】処理ユニット１１０はＡＬＵでもシフタでもＭＡＣでもよい。記憶ユニット１００は一般に何らかの種類のメモリであってよく、レジスタファイルでも、半導体メモリでも、磁気メモリでも、またはいくつかの種類のメモリのいずれのものでもよい。処理ユニット１１０は加算、減算、論理ＡＮＤ、論理ＯＲ、バレルシフタでのようなシフト、乗算、累算、およびデジタル信号プロセッサにおいて典型的に見られる乗算および累算のような、典型的な演算を行なってもよい。処理ユニット１１０はオペランドを、命令において用いられる１つのオペランド、命令において用いられる２つのオペランド、またはそれ以上多くのものなどのうちいずれかとしてとる。処理ユニット１１０は次にこれらのオペランドで演算を行なって、それらの結果を得る。スカラまたはベクトルオペランドで開始することにより、オペランドは演算を最後まで行なわれ、それぞれスカラまたはベクトル結果をもたらす。

【００３３】次のステップは、処理ユニット１１０がどのように形成されてもよく、どのように機能するかをより特定的に認識するためのものである。データおよびプログラムは記憶ユニット１００内で組合せられているように示されているが、それらは同じ物理的メモリ内で組合せることもできるし、別個になった物理的メモリ内で実現することもできるということは明らかであろう。各オペランドは典型的な３２ビットの長さを有するものとして説明されているが、一般に、オペランドはいくつかの長さのいずれとすることもできるだろう。１６ビットマシン、８ビットマシン、または６４ビットマシン等々とすることができる。一般的なアプローチは、Ｎビットオペランドが、ともにとられて加算されるとＮビットになる複数オペランドとして考えられ得るということであると、当業者は認識するだろう。したがって、３２ビットワードはたとえば２つの１６ビットハーフワード、もしくは４つの８ビットクォーターワードまたはバイトであり得るだろう。発明者らによる現在の実現例では、１つのオペランド中の要素は各々同じ幅のものとしている。しかしながら、３２ビットオペランドを、一方の要素を２４ビットとし、他方の要素を８ビットとすることもできる。オペランド中で複数のデータ経路および複数の要素を用いることから導き出される利点とは、すべての要素が独立的かつ同時に処理されており、処理のスループットの増加がなし遂げられるということである。

【００３４】命令はどのようなサイズであってもよい。現在は３２ビット命令が用いられている。しかしながら当業者は、８ビット、１６ビット、３２ビット、および６４ビットにおいて特に有用性を見出すかもしれない。より重要なことは、命令については固定長でさえなくともよいということである。同じコンセプトが、３２ビット命令に拡張可能な１６ビット命令を備えるものなどの、または命令がいくつかの数の８ビットバイトで形成されており、その数はそれがどの特定の命令であるかによって決まる、可変長命令マシンにおいて用いられた場合でも、働くだろう。当業者のために、図２８～３６に例示的な命令セットのまとめを示し、本発明に従って実現されてもよい命令を示す。

【００３５】処理ユニット１１０は典型的にはＡＬＵ１２１および／またはＭＡＣ１２２を含んでもよい。またこれは、シフタ１２３または論理ユニット１２４を実現するだけのものであってもよい。

【００３６】加算器

図３は処理ユニット（図２の１１０）のための、ＡＬＵにおいて実現されてもよい加算器を模式的に表わしたものである。図３（ａ）は従来の３２ビット加算器を示す。図３（ｂ）はハーフワード対モードのために接続された２つの１６ビット加算器を表わしたものである。図３（ｃ）はワードモードのために接続された２つの１６ビット加算器を表わしたものである。

【００３７】図３（ａ）から（ｃ）は、図３（ａ）における３２ビットの従来のマシンにおける典型的なハードウェアが、本発明に従うハーフワード対モードまたはワードモードの所望される目的を達成するためにどのように変形されてもよいかということを示す役割を果たす。ベクトルはここでは２つの要素を持つものとして示される。より特定的には、３２ビットの従来のオペランドがどのようにして各々１６ビットの２つの要素に分割され得るかということが示される。同じ原理を、等しい長さまたは等しくない長さのものがあるいくつかの要素に分割するのに適用することができるだろう。

【００３８】図３（ａ）を参照して、従来の加算器２００はＸオペランドのための入力ＸとＹオペランドのための入力Ｙとを有する。またこれは、加算器と関連して典型的に見出されるキャリーーイン２０１および条件コード２０５のための入力をも有する。条件コード２０５は、オーバフローを表わすのがＶ、キャリーーアウトがＣ、ゼロ結果、すなわち加算器から出される結果がゼロである場合がＺであってよい。さらにこれは加算器から出される結果オペランドを有しており、これはＳである。Ｘ、Ｙ、およびＳはすべて３２ビットワードで表わされる。制御入力ｓ／ｕ２０２は符号付または符号なしオペランドを表わし、ここで最上位ビットはその数が正

または負である場所を示し、もしくは符号なしオペランドではその最上位ビットがオペランドの大きさに関与する。図３（ｂ）は、典型的な３２ビット加算器に類似してはいるが、そうではなく単なる１６ビット加算器である２つの加算器が、どのようにともに組合せられてハーフワード対、すなわち１つのオペランドにつき２つのハーフワード要素があるものに対してベクトル演算を行なうことができるかということを示す。Ｙオペランドはここでは２つのハーフワードオペランド、すなわち下半分のＹ０からＹ１５、および上半分のＶ０からＶ１５として分割されている。同様に、Ｘオペランドは２つのハーフワードオペランド、すなわち下半分のＸ０からＸ１５、および上半分のＵ０からＵ１５として分割されている。結果Ｓは、加算器２１０からくるＳ０からＳ１５、および加算器２２０からくる上半分のＷ０からＷ１５として認識される。本質的には、３２ビット加算器２００を中央で分割して２つの１６ビット加算器２１０および２２０を形成してもよい。しかしながら、上位ビットにはオペランドの符号ビットの性質を決定するための論理が必要であろう。したがって３２ビット加算器２００を分割する際には、３２ビット加算器から分割されて加算器２１０を形成する下方の１６ビットの符号制御のために付加的な論理が必要となるであろう。この場合これら２つの加算器２１０および２２０は、加算器２１０のための入力オペランドが３２ビットオペランドの下半分からきており、１６ビット加算器２２０のための入力オペランドが３２ビットオペランドの上半分からきているということを除けば、同一なものとなるであろう。

【００３９】オペランド要素ＸおよびＵが別個にそれぞれＹおよびＶと加算されて合された場合、それらはそれぞれ結果ＳおよびＷをもたらす。またそれらは加算器の各々のために独立的な条件コードを生成する。加算器２１０は条件コード２１５を生成し、加算器２２０は条件コード２２５を生成する。これらの条件コードは、それらが関連している特定のハーフワード加算器に適用される。したがって、これで独立的なハーフワード対演算を行なうために従来の３２ビット加算器がわずかに変形される様が見てとれる。

【００４０】図３（ｃ）を参照して、図３（ｂ）における同じ加算器ユニットが、図３（ａ）の加算器２００において行なわれた、もとのワード演算を行なうべく再接続されてもよい。これは、オペランドが３２ビットスカラを表わす場合である。スカラはＹ０からＹ３１およびＸ０からＸ３１である。これらのオペランドの下半分は加算器２３０によって処理され、上半分は加算器２４０によって処理される。これを可能にするメカニズムは、加算器２３０のキャリーアウトを加算器２４０のキャリーイン２３６に接続することによるものである。図２（ｃ）に示されるように、組合せられた２つの１６ビット加算器は図３（ａ）の１つの３２ビット加算器と同じ機能を果たす。したがって、図３（ｂ）および３（ｃ）に示した実現例では、加算器２１０は本質的に加算器２３０と同じものであってもよく、一方で加算器２２０は加算器２４０と同じものであってもよい。この説明ではこれら２つの加算器がどのようにハーフワード対モードまたはワードモードのいずれかで機能できるかが示されているが、当業者は、拡張によってベクトルの独立した要素を同時に扱うために従来の加算器をいくつかの加算器に変形すること、およびこれを再結合してスカラ演算をスカラオペランドで行なうことをしてもよい。

【００４１】図３の加算器について、１つ注目すべきことがある。図３（ｃ）では２組の条件コード２３５および２４５が示されている。一方、もとの従来の加算器では１組の条件コード２０５しかない。図３（ｃ）の条件コードは、本当は条件コードＺを除いては２４５の条件コードである。２３５における条件コード、すなわちオーバフローＶおよびキャリーＣは、条件コード２０５における条件コードおよび条件コードＺが、効果的に２３５のＺ条件コードとＡＮＤ処理される２４５のＺ条件コードである限り、無視される。ここでは２０５の条件コードＶは２４５のＶに対応する。２０５のＣは２４５のＣに対応し、２０５のＺはコード２３５のＺとＡＮＤ処理されたコード２４５のＺに対応する。当業者は、適合すると思われるどの特定のやり方でもこれらを組合せることができるだろう。

【００４２】論理ユニット

図４は本発明に従い実現されてもよい論理ユニットの模式図である。図４（ａ）はビット単位の論理演算、ビット単位の補数または現在のプロセッサにおいて典型的に見られるいくつかの組合せを行なう典型的な３２ビット論理ユニットを示すものであって、これらの演算について重要かもしれないのは、それらが条件コードにおける異なったビットのために独立的に働くということである。オーバフロービットは通常、３０５における条件コードでは全く重要性を持たない。キャリーアウトは論理演算においてまったく重要ではないが、ゼロには、結果がゼロであるということを示すことにおいてまだ重要性がある。ハーフワード対演算のために、もとの３２ビット加算器は「動作的」には２つの１６ビット論理ユニットに分割されるだろう。入力オペランドにおける上方の１６ビット３２０および下方の１６ビット３１０は、加算器のときと同じ態様で２つのハーフワードに分割されるだろう。論理演算を処理するにあたっては、ビットは一般に独立的に処理されるので、２つの論理ユニット３１０および３２０の間には動作的な接続は全くない。

【００４３】図４（ｃ）はスカラ処理のための典型的な論理ユニットを形成するようにもう一度再結合された論理ユニットを示す。条件コードエリア以外ではユニット間には接続が必要ではないということに注意されたい。従来の論理ユニットのゼロ条件コード３０５はここでは

ユニット３４５のゼロ条件コードをユニット３３５のゼロ条件コードとＡＮＤ処理することによって表わされてもよい。したがって当業者には、デュアルモード論理ユニットが前述のようにデュアルモード加算器のコンセプトおよび実現例を拡張することによって構成され得るということが明らかなはずである。

【００４４】シフタ

図５から８は、本発明に従い実現されてもよいバレルシフタを模式的に表わしたものである。いくつかのプロセッサは図５（ｂ）に示されるようにバレルシフタを有するが、他のものは図５（ａ）、図６、および図７に示される１ビットシフタを有する。バレルシフタは典型的にはプロセッサユニット内に必要なものではないが、高性能マシンについては、プロセッサユニットは図５（ｂ）に表わされるようなシフタユニットを実現してもよい。以下の説明では、処理を高速化する、または必要なハードウェアの量を最小限にするために、当業者によってシフタがどのように構成され実現されてもよいかを示す。図５（ａ）は、左シフトまたは右シフトのどちらかである１ビットシフトが典型的なプロセッサにおいてどのように実現されてもよいかを示す。シフタ４１５は、３２ビット入力オペランドＸが、左または右へ１ビットシフトされる、または方向入力ＤＩＲ４０１の制御下ではシフトされないようにして、Ｚ出力を生成できる。シフトが起こった場合、それが左へのシフトなら、選択ボックス４１６によって最下位ビットの位置にビットが入れられなければならない。

【００４５】シフタが右へシフトされる場合、選択ボックス４００からのビットが最上位ビットの位置に入れられる。選択ボックス４００および４１６はシフタ４１５に入れるために選択され得るいくつかの入力を有する。双方のボックスにはＳＥＬとラベル付けされる選択入力もあり、これは命令からくるものであって、従来のマシンには典型的なものである。ＳＥＬはこれらの入力ビットのうちどちらがシフタに入れられるために選択されるであろうかを決定する。一般に、これらの選択ボックスがあるため、シフトは、シフタの外へシフトされるビットがシフタのもう一方の端で中にシフトされる回転でもあり得るし、他のビットが右へシフトされる際に符号ビットまたは最上位ビットがドラッグされる算術的右シフトでもあり得るし、他のビットが左へシフトされる際に０が入れられる算術的左シフトでもあり得る。論理シフトについては、「０」がビットとして入れられる。また、「１」は論理シフトに入れられる新しいビットとして、入れられる。

【００４６】当業者は、加算器および論理ユニットのための条件コードの説明を参照することによって、容易に条件コードをシフタに割当て、算術的左シフト演算のためのオーバフロー、シフト演算の最後のビットを保持するためのキャリー、およびシフトの結果が０値であったときにそれを記録するゼロフラグを表わすことができるだろう。

【００４７】図５（ａ）のシフタを組合せて用いることで、図５（ｂ）のシフタを３２ビット左／右バレルシフタとして形成してもよい。これは図５（ａ）におけるシフタを３２個組合せ、それらを次々にカスケード接続し、第１のものの出力が第２のものの入力に入る、というふうに最後まで続いていくようにすることによって行なわれてもよい。シフトされるべきビットの数は個々のシフタへの方向入力ＤＩＲの１および０のパターンによって決定される。図５（ａ）ではシフタのための方向は３値であるということに注意されたい。すなわち左、右、またはまったくシフトがなし遂げられない真っ直ぐ前方、である。そこで図５（ｂ）では、個々の３２ビットの１ビットシフタへの方向入力は、左でも右でもシフトなしでもあり得る。３２ビットが左へシフトすべきである場合、すべての方向入力が左を示すだろう。

【００４８】左へシフトすべきなのが１ビットだけの場合、第１のボックスが左への１ビットシフトを示し、他の３１個はすべてシフトなしを示す。Ｎビットが左へシフトすべきである場合、始めのＮ個のボックスが左への１ビットの方向入力を有し、残りのボックスがシフトなしを示すだろう。同じことが右へのシフトにも適用できるだろう。この場合には方向は右へのシフトまたはシフトなしのいずれかを示し、同じように右シフトにおいて０ビットから３２ビットまでのシフトが可能であろう。

【００４９】この図５（ａ）における典型的な１ビットシフタは、ここで図６を参照して２つの１６ビットシフタに分割することができる。ここではハーフワード対モードのために接続された２つの１６ビットＬ／Ｒ１ビットシフタが示される。図５（ａ）におけるシフタ４１５は、動作的には２つの１６ビットの１ビットシフタ４５０および４３５に分割できる。これらの１６ビットシフタの各々は、この場合特に４１６および４００を参照する図５（ａ）に示す入力選択論理を有しており、これはボックス４５０がボックス４６０および４４５を有し、ボックス４３５がボックス４４０および４３０を有するように二重にされる。入力論理は同じであるが、選択ボックスへの入力は異なったように結線される。したがって、ハーフワード対モードのために接続される図６のシフタとワードモードのために接続される図７のシフタとの違いは、入力選択ボックスの結線のされ方にある。下方のシフタ４５０のための図６の入力オペランド要素はＸ０からＸ１５であり、シフタ４３５のための入力オペランド要素はＹ０からＹ１５である。このようにしてＸおよびＹは２つのハーフワードを示す。

【００５０】結果Ｚ出力オペランドは２つのハーフワードとして示される。下方の１６ビットはＺ０からＺ１５であり、上方の１６ビットはＷ０からＷ１５である。入力セレクタは、回転においてシフタから出力されるビッ

トがシフタの他方の端にフィードバックされるように結線される。シフタ４３５が左シフトを行なうと、回転されるビットはＹ１５となり、右シフトを行なうと回転されるビットはＹ０となる。同様にシフタ４５０について、それが左回転であれば、入力ビットはＸ１５であり、右回転であれば入力ビットはＸ０である。同様に、選択は算術的シフトおよび論理的シフトについても図５（ａ）でのように働く。

【００５１】図７は、これらの同じ２つのシフタの動作がどのようにワードモードのために接続され得るかを示す。ここではシフトパターンは図６での２つのハーフワードとは違ってオペランドにおける３２ビット全体に対して働く。左への回転については、下方のシフタ４８６から外へ回転させられるビット（ＭＳＢビットＸ１５）は上方のシフタ４７５の中に回転させられる一方で、ＬＳＢビットはシフタ４７５に入力される。これは上方の１ビットシフタと下方の１ビットシフタとの間で連続的なシフトを形成する。２つのシフタをめぐる回転については、Ｘ３１がＸ０にシフトされるだろう。図７に示したようにセレクタ４８０のすべての入力がＸ１５に接続されており、セレクタ４８５のすべての入力がＸ１６に接続されていれば、図７の組合せられたシフタは図５（ａ）におけるシフタとして効果的に動作する。入力セレクタ４７０は入力セレクタ４００と同じパターンを有するだろう。入力セレクタ４８８はセレクタ４１６と同じ入力パターンを有するだろう。したがって、図７の組合せられたシフタは図５（ａ）におけるシフタと同じスカラオペランドのためのシフト動作を行なうだろう。

【００５２】図６および７における１ビットシフタはさらに、１ビットシフタを３２個カスケード接続することによって、図５（ｂ）と類似の態様で、図８に示した３２ビットバレルシフタに拡張することができる。１ビットシフトが所望されるならば、方向制御信号が第１のシフタに対して用いられ、１ビットシフトを示す。他のカスケード接続された１ビットシフタに対しては、示されるシフトはない。Ｎビットシフトについては、最初のＮ個の１ビットシフタにおける方向入力が、１ビットだけシフトすることを示し、残りの１ビットシフタはシフトせずデータを通過させる。

【００５３】同様にこの図８のバレルシフタはワードまたはハーフワード対モード演算のいずれをも行なうことができる。なぜなら、個々のビットシフタはワードまたはハーフワード対演算のどちらでも行なうことができるからである。この図５から８の実施例はバレルシフタを実現する１つの方法を代表するものであるが、バレルシフタを真中で分割して入力選択論理を提供する同じコンセプトが、バレルシフタの多くの他の実現例にも応用できる。当業者は、特定のハードウェアまたはスループットの要求に応じて適切な実現例を見出すことができるはずである。

【００５４】乗算累算器

図９および１０は、本発明に従い実現されてもよい乗算および累算（ＭＡＣ）ユニットの模式図である。

【００５５】典型的な３２ビットプロセッサは通常、高価な３２×３２乗算器アレイの実現を必要とはしないだろう。乗算はおそらく他の方法で確立されるだろう。しかしながら典型的な１６ビット信号プロセッサでは、１６×１６乗算器アレイが極めて普通に見られる。高速な乗算を必要とするタイプの計算には、典型的に１６ビットデータが用いられるので、１６×１６乗算器アレイの方が普及したものとなっており、これはいくつかの３２ビットプロセッサにおいてさえ当てはまることである。したがって、３２ビットオペランドを２つの１６ビットハーフワード対として扱うことにより、１つのベクトル化されたオペランド中の３２ビットワードオペランド、ハーフワードオペランド、またはハーフワード要素の空間ベクトルの概念を利用すべく２つの１６×１６乗算器アレイを実現することができる。

【００５６】次の例は、どのようにして１６×１６乗算器アレイを二重にして２つのハーフワード対乗算器として用いることができるかを示すものであって、これらの乗算器はともに接続されて３２×１６スケーラ乗算をもたらしてもよい。この３２×１６スカラ乗算には、これらの乗数の２つを一緒に用いて３２×３２ビット乗算をなすことができるという有用さがある。または、３２×１６乗算をそれ自体で用いることもでき、この場合３２ビットの精度のオペランドがただ１６ビットだけの精度のオペランドによって乗算されてもよい。

【００５７】ＭＡＣユニットはすべてのプロセッサで典型的に見られるわけではない。しかし信号処理の用途のための高性能プロセッサでは、これは典型的に実現されている。図９は、ＭＡＣユニットの従来の実現例を示す。ＭＡＣは様々なサイズのうちどのサイズでもあり得る。これは３２ビットの積を形成する乗算器における１６ビット×１６ビットのユニットである。この３２ビットの積は累算加算器内で第３のオペランドと加算されてもよく、これは「ガードビット」と呼ばれる余剰の上位ビットがあるためその積よりも長いかもしれない。

【００５８】図９に示されるように、入力オペランドは１６ビットであって、Ｘ０からＸ１５およびＹ０からＹ１５で表わされる。これらは３２ビットの積Ｚを発生し、これはフィードバックオペランドＦに加えられてもよい。この場合、Ｆは４０ビットのフィードバックワードまたはオペランドを表わすＦ０からＦ３９として示される。これが４０ビットなのは、積を保持するのに３２ビット、加えてガードビットのために付加的な８ビットが必要とされるであろうからである。ガードビットはオーバフローを扱うために含まれている。なぜなら、いくつかの積が加算されると、オーバフローが起こる可能性があり、ガードビットはオーバフローを累算してそれら

を保護するからである。典型的にはガードビットの数は４または８であろう。この例では８ビットが示されているが、いくつかのサイズが可能であろう。累算器の結果は４０ビットの結果Ａ０からＡ３９として示される。

【００５９】乗算アレイは乗算器なしで用いることもできるし、乗算器とともに用いることもできるということに注意すべきである。符号付または符号なしを意味する別の入力Ｓ／Ｕが、入力オペランドが符号付数として扱われるべきか符号なし数として扱われるべきかを示すということが注意されるべきである。当業者は、乗算器の上方のビットが、入力オペランドが符号付であるか符号なしであるかによって異なったように扱われるということを認識するであろう。

【００６０】図１０は、ハーフワード対オペランドを扱うためにどのように典型的な１６×１６アレイが形成されるかを示す。この場合、３２ビット入力オペランドＸが２つのハーフワードに分割される。乗算器５２０のための下方のハーフワードはＸ０からＸ１５であり、乗算器５１５のための上方のハーフワードはＸ１６からＸ３１である。Ｙ入力オペランドも２つのハーフワードオペランドに分割される。乗算器５２０のための下方のハーフワードはＹ０からＹ１５であり、乗算器５１５のための上半分はＹ１６からＹ３１である。図１０はこのようにしてＸオペランドのハーフワードオペランドをそれぞれＹオペランドのハーフワードオペランドと乗算するための接続を表わす。Ｘの最下位ハーフワードは乗算器５２０においてＹの最下位ハーフワードと乗算されるということに注意されたい。また乗算器５１５において独立的かつ同時に、Ｘの上方のハーフワードがＹの上方のハーフワードと乗算される。これらの２つの乗算は、２つの積を生じる。乗算器５２０からの３２ビットの積はＺ０からＺ３１で表わされ、同様に乗算器５１５の３２ビットの結果はＷ０からＷ３１によって表わされる。２つの積はその精度を保つために各々１６ビットよりも大きい。この時点で、ハーフワードの積は独立的なオペランドの表現として保存される。

【００６１】乗算器５２０より出された下方のハーフワードからの積は、累算器５３０に送られＦ０からＦ３９で表わされるフィードバックレジスタで加算される。これにより、Ａ０からＡ３９で表わされる累積された積Ａが形成される。同様に上方のハーフワードにおいて、積はＷ０からＷ３１によって表わされており、かつ累算器５２５の中でＧ０からＧ３９によって表わされるフィードバックレジスタに加算されて４０ビットの結果Ｂを形成し、この結果はＢ０からＢ３９で表わされる。これらの累算器の結果は一般に、乗算の精度を保つためにより大きい数またはビットで表わされるオペランドとして累算器の中でより大きい数として保存される。

【００６２】フィードバックビットは通常、メモリ（図２の１００）またはより大きい数のビットをストアすることのできる特殊なメモリのいずれからでももたらされるだろう。典型的なメモリ位置が扱えるのは３２ビットであるが、典型的には累算器ファイルと呼ばれる特殊なメモリは、スカラ積のために４０ビット、またはハーフワード対の積のために８０ビットをストアすることができるだろう。この場合スカラオペランドを扱うことのできる２つの累算レジスタが、ハーフワード対オペランドのための記憶を形成するのに用いられてもよい。換言すれば、ハーフワード対演算の２つの４０ビットの結果をストアするのに２つの４０ビット累算器を用いることができるだろう。

【００６３】ＭＡＣ相互接続

図１１および図１２は、スカラオペランドのための１６×３２ビット乗算を形成するために、図１０のアレイの２つの１６ビット乗算器がどのように相互接続され得るかを示す。この例では、乗算器アレイは加算器列として実現される。最下位乗算器アレイ６１０のキャリーアウト６０５は、上位乗算器アレイ６００の加算器にキャリー入力として与えられる。さらに、上位乗算器アレイ６００の最下位端に形成される合計ビット６０６は、下位乗算器アレイ６１０の加算器の最上位端に与えられる。

【００６４】他の接続は、累算器６１５および６０５に生ずる。積の下位部分を表わす累算器６１５は３２ビットに制限され、上位８ガードビットは使用されない。３２ビットのキャリーアウトは上位４０ビット累算器６０５のキャリー入力に与えられ、その結果はＢ３９を通るＢ０としてここでは示される７２ビットオペランドである。典型的にはこのオペランドは２つのオペランドとしてストアされ、下位３２ビットは１つの累算器６１５にストアされ、上位４０ビットは第２の累算器６０５にストアされる。さらにこの演算では、符号付ビットおよび符号なしビットのために、入力オペランドＸの下位半分は乗算器６１０において符号なし数として扱われ、入力オペランドＸの上位１６ビットは上位乗算器アレイ６００において符号付または符号なしオペランドとして扱われる。

【００６５】さらに、下位累算器６１５においては積は符号なしオペランドとして扱われ、一方上位累算器６０５ではオペランドは符号付数として扱われる。４０ビット累算器は図１１および図１２のすべての例においては符号付数として扱われることを付け加えるべきである。これは、符号なし数でさえも符号付数の正の部分と考えられ得るようなビットを、累算器の拡張であるガードビットが可能にするからである。ゆえに、拡張累算器における符号付数は、符号付オペランドと符号なしオペランドの両方を含む。

【００６６】図１２は、乗算器アレイ６００および６１０を構成する加算器間でキャリーおよび合計ビットがどのように相互作用するかをより詳細に示す。たとえば、

加算器６２５および６３５は乗算器アレイ６１０の一部として示され、加算器６２０および６３０は乗算器アレイ６００の一部として示される。乗算器アレイ６１０および６００は典型的には加算器の何らかの構成でもって実現されることが注目されるべきである。特定的な実現例において、加算器の相互接続は様々な方法でなされるであろう。図１０は加算器の単純なカスケードを示すが、この同じ技術を、加算器がたとえばブース乗算器またはウォレス・ツリー乗算器におけるように接続されるであろうような他の方法に用いてもよい。図１２に示されるように、下位乗算器アレイ６１０の加算器６２５は、上位乗算器アレイ６１０の対応する加算器６２０のキャリー入力に与えられるキャリーアウト６２１を与える。下位乗算器アレイ６１０は、Ｘ－入力の入力オペランドがあたかも符号なしであるかのように演算を行なう。入力オペランドの符号は特定されて、上位乗算器６００アレイの上位加算器６２０および６３０の符号制御に用いられる。

【００６７】さらに加算器は、それらが乗算器の最下位ビットから乗算器の最上位ビットにオフセットされるような方法で接続されるため、それは合計ビットを再び加算し戻す機会を与える。より特定的には、加算器６２５および６２０は、Ｙｉとされる、乗算器のより下位のビットに対応する。加算器６３５および６３０は、Ｙ（ｉ＋１）とされる、乗算器の次のより上位のビットに対応する。このオフセットは、加算器６３５の入力Ｂ０に与えられる加算器６２５の出力Ｓ１、および加算器６３５の入力Ｂ１４に与えられる加算器６２５のＳ１５として見られ得る。この１ビットのオフセットは加算器６２０からの入力Ｓ０を受取るよう加算器６３５の入力のＢ１５を解放して、最上位乗算器アレイ６００からの合計ビットは最下位乗算器アレイ６１０へ入力ビットとして与えられる。

【００６８】さらに、加算器６２５からの合計ビットＳ０は、６４０として示される次の部分積に直接進み、さらなる乗算器または加算器段を通る必要はない。したがって、連続する加算器段６２５、６３５などからＳ０を出力することは、図９の出力ビットＺ０からＺ１５を生じさせる。加算器６３５のＳ０からＳ１５に対応する最終部分積からの出力ビットは、Ｚ１６からＺ３１の図９のアレイ６１０からの出力ビットを生じさせるであろう。

【００６９】万一乗算器Ｙが負である場合に補償を与えるために最終加算器段がどのように用いられ得るかについては、当業者ならば理解するであろう。

【００７０】<u>オペランドデータのタイプの分類</u>

オペランドデータのタイプの分類に関してここで注目する。オペランドモードのタイプをスカラまたはベクトルとして特定するための１つのアプローチは命令にその情報を含むことであるが、代替的アプローチはその情報をオペランドの付加的なビットにおいて付け加えることである。たとえば、オペランドが３２ビットの場合、１つの付加的なビットを用いて、オペランドをスカラまたはベクトルのいずれかとして識別してもよい。仮にベクトル要素の数がはっきりと示されるか、またはベクトル要素の数が２のような何らかの数であると仮定され得るような場合、付加的なビットがさらに用いられてもよい。オペランド処理ユニットは、オペランドに付加される情報に応答することによってオペランドをスカラとしてまたはベクトルとして処理するのに適合されるであろう。

【００７１】オペランドがスカラであるかまたはベクトルであるかは、オペランドが選択される方法によってさらに特定されてもよい。たとえば、オペランドのアドレスをさらに特定するメモリ位置にあるビットフィールドに、情報が含まれてもよい。

【００７２】２つのオペランドが処理ユニットによって処理され、モード情報がその２つのオペランドにおいて異なる場合には、混合されたモード演算を処理するために当業者によって処理ユニットに規定が設定されてもよい。たとえば、ベクトルオペランドおよびスカラオペランドを伴うＡＤＤ演算は、処理ユニットによって、スカラからベクトルを形成し、必要ならば切捨て、次いでベクトル演算を行なうことによって処理されてもよい。

【００７３】<u>空間ハードウェアに対する代替としてのタイムシェアリング</u>

実現手段をタイムシェアリングすることは空間に分散する実現手段の代用にしばしばなり得ることは、当業者には理解されるであろう。たとえば、空間に分散されるベクトル処理ユニットにおいて多重加算器を効果的に実現するのに、１つのベクトル加算器が何度も用いられてもよい。ハードウェアの多重化および非多重化を用いて、入力オペランドおよび結果を順序づけることも可能である。付加的なサポートハードウェアを有するベクトル加算器をさらに用いて、スカラオペランドを処理するのに分散ベクトル加算器が相互接続され得る方法と類似の態様でスカラオペランドをばらばらに処理することも可能である。サポートハードウェアは、ベクトル演算処理素子間を通る中間結果を処理するのに用いられる。

【００７４】この発明の上記の説明に留意して、この発明の空間ベクトルデータ経路を組込む例示のＲＩＳＣ型プロセッサがこれより説明される。以下のプロセッサシステムは、当業者がこの発明を組込むであろう方法の一例にすぎないことに注意されたい。他の例は、記載されるこの発明に基づく、それらの有利なアプリケーションを見出すであろう。

【００７５】<u>この発明を組込む例示的プロセッサ</u>

この発明を組込む演算処理素子の機能図を示す図１３を参照する。以下の説明は特定のビット幅を参照するが、それらは例示のためであり、この発明の教示に従って他の幅が容易に構成され得ることを、当業者は理解するで

あろう。

【００７６】図１３を参照すると、図示されるデータ処理ユニットを制御するために、２つのソースオペランドおよび１つの宛先オペランドを特定することのできる命令が用いられる。

【００７７】オペランドは典型的にはレジスタにおよびデータメモリ（２００）にストアされる。演算命令、論理命令、およびシフト命令がＡＬＵ２４０およびＭＡＣ２３０においてレジスタ空間からのオペランドを用いて実行され、その結果はレジスタ空間に戻される。レジスタ空間はレジスタファイル２２０と幾つかの他の内部レジスタ（図示せず）とから構成される。レジスタ空間にストアされるオペランドは、３２ビットワードまたはハーフワード対のいずれかである。オペランドは、ロードおよびストア命令によってレジスタ空間とメモリ２００との間を、または既に記載したようにレジスタ空間と自動メモリアクセスユニットであるストリーマ２１０との間を往復する。

【００７８】図１４を参照すると、ＡＬＵ２４０の機能ブロック図が示される。ＡＬＵは加算器４１０、４２０とバレルシフタ４７０とから構成される。一般に、ＡＬＵ命令は、レジスタ空間から２つのオペランドをとり、レジスタ空間にその結果を書込む。ＡＬＵ命令は、各クロックサイクルを実行することができ、ＡＬＵパイプにおいて僅か１つの命令クロックサイクルを必要とするだけである。

【００７９】加算器４１０、４２０およびシフタ４７０は、ワードまたはハーフワード対オペランドを用いて演算を行なう。符号付オペランドは２の補数表記法で表わされる。現在、符号付、符号なし、小数、および整数オペランドが、ＡＬＵ演算のための命令によって特定可能である。

【００８０】<u>加算器</u>

加算器（４１０、４２０）はワードおよびハーフワード対で加算および論理演算を行なう。ハーフワード対演算の場合、加算器４１０、４２０は半分のものが２つあるものとして機能する。下半分４２０はハーフワード対の下位オペランド４６０を用いて演算を実行し、上半分４１０は同じ演算をハーフワード対の上位オペランド４５０を用いて実行する。ハーフワード対モードにある場合は、２つの加算器４１０、４２０は本質的に互いから独立している。３２ビット論理ユニット４４０は、下の加算器４２０から上の加算器４１０へ情報を送り、２つの加算器がワードモードで動作しているときには情報を逆に送るために用いられる。

【００８１】加算器演算は、２つのキャリー（ＣＵおよびＣＬ）、２つのオーバフロー（ＶＵおよびＶＬ）、および２つのゼロ（ＺＵおよびＺＬ）条件コードビットに影響する。ＣＵはワード演算のための桁上げフラグであり、ＣＵおよびＣＬはハーフワード対演算のための桁上げフラグである。同様に、ＶＵはワード演算におけるオーバフローを示し、ＶＵおよびＶＬはハーフワード対演算におけるオーバフローを示す。

【００８２】オーバフローフラグに作用するオーバフローは、加算器演算命令からおよびＭＡＣスカラ命令から結果として生じ得る。オーバフローフラグは、実行された命令がたとえ結果を飽和したとしてもセットされる。一度セットされると、条件コードは、フラグをセットすることのできる別の命令があるまで変わらない。

【００８３】飽和のない加算器演算命令がオーバフローし、誤り例外が可能化されると、誤り例外要求が生ずる。飽和のあるオーバフローおよび飽和のないオーバフローを示すために、別個の信号がデバッグ論理に送られる。

【００８４】<u>バレルシフタ</u>

図１４を参照すると、１クロックサイクルの間に、バレルシフタは３２ビット位置までのワードオペランドにあるすべてのビットを左または右のいずれにもシフトさせながら、ゼロ、オペランドの符号ビット、または加算器の上位桁上げフラグ（ＣＵ）を回転または挿入することができる。ハーフワード対演算の場合には、１クロックサイクルで、シフタは１６ビット位置までの両方のハーフワードを左または右へシフトさせながら、ゼロ、符号ビット、または加算器の桁上げフラグ（ＣＵおよびＣＬ）を回転または挿入することができる。

【００８５】典型的なシフト／回転演算の場合、バレルシフタ４７０は、両方のソースオペランドの位置にある各ビットを演算によって示される方向に移動させる。各位置のシフトに対して、バレルシフタ４７０は、選択される演算に依って、終わりのビットを回転させるか、または符号ビット、桁上げフラグ（ＣＵまたはＣＬ）、もしくはゼロを挿入する。

【００８６】たとえば、左回転の場合、ビットは左側へシフトされる。ビット３１はワードモードではビット０にシフトされる。ハーフワード対モードの場合には、ビット３１はビット１６に回転させられ、ビット１５はビット０に回転させられる。右回転の場合は、ビットは右側にシフトされる。ゼロはワードモードではビット３１に挿入される。ハーフワード対モードの場合には、ゼロはビット３１およびビット１５の両方に挿入される。同様に、キャリー伝搬を伴うシフトでは、桁上げフラグ（ＣＵ）はワードモードではビット３１に挿入される。ハーフワード対モードの場合には、各ハーフワードの桁上げフラグ（ＣＵおよびＣＬ）はビット３１およびビット１５に挿入される。

【００８７】次に図１５を参照する。デュアルＭＡＣユニットは、２つの１６×１６の積または１６×３２の積のいずれをも生ずることができるよう一体的に相互接続された、２つのＭＡＣユニット５２０、５５０、５７０、５９０および５１０、５４０、５６０、５８０から

構成される。各ＭＡＣは、１６×１６乗算アレイ５１０、５２０と、累算加算器５６０、５７０と、累算器レジスタファイル５８０、５９０と、スケーラ５９１とから構成される。

【００８８】幾つかの例示的な命令：乗算、累算、乗算および累算、ユニバーサルハーフワード対乗算、ユニバーサルハーフワード対乗算および累算、ダブル乗算ステップ、ならびにダブル乗算および累算ステップが、図２８－図３６に挙げられる命令のまとめに見られる。

【００８９】ワード演算はどちらかのＭＡＣユニットで実行され得る。ＭＡＣは現在１６×１６演算であるため、ＭＡＣユニットで用いられる「ワード」は１６ビットであることは注目されるべきである。しかしながら、より便利なアプローチは、ベクトル長１、２、４または８を用いて演算を表わすことである。したがって、ＭＡＣにおけるワード演算はベクトル長１と呼ばれることができ、一方ハーフワード対演算はベクトル長２となるだろう。宛先累算器を含むＭＡＣは、演算を行なうのに現在用いられているものである。

【００９０】ハーフワード対演算は両方のＭＡＣユニットを用いる。命令は特定の累算器を宛先累算器として特定し、これはアドレス指定される累算器となる。アドレス指定される宛先累算器を含むＭＡＣは下位のハーフワード対要素で演算を行ない、他方の（「対応する」）ＭＡＣは同じ演算を上位のハーフワード対要素で行なう。対応するＭＡＣからの結果は対応する累算器にストアされ、アドレス指定される累算器と対応する累算器とはそれらのそれぞれのレジスタファイルにおいて同じ相対位置に位置する。

【００９１】倍精度演算はハーフワードおよびワードで行なわれ、この演算は二重ＭＡＣとして組合せられる２つのＭＡＣによって行なわれる。「上位」ＭＡＣは計算の最上位部を行ない、「下位」ＭＡＣは計算の最下位部を行なう。

【００９２】ＭＡＣユニットは、整数オペランドまたは小数オペランド、および符号付または符号なしオペランドをサポートしてもよい。

【００９３】累算器レジスタファイル

２つのＭＡＣユニットは上位ＭＡＣおよび下位ＭＡＣと呼ばれる。各ＭＡＣは４つの４０ビットのガードされる累算器レジスタから構成される累算器レジスタファイルを有し、ＡＬＵには合計８つの累算器がある。各ガードされる累算器（ＡＧｎ）は、最上位端が８ビットのガードレジスタ（Ｇｎ）でもって拡張される３２ビット累算器レジスタ（Ａｎ）から構成される。図１６は累算器レジスタファイルのレイアウトを示す。

【００９４】ハーフワード対オペランドの累算器は２つの累算器にストアされる。ハーフワード対の下位要素は、いずれかのＭＡＣの１つの累算器において、４０ビット数として累算される。ハーフワード対の上位要素は、他方のＭＡＣにある対応する累算器において、４０ビット数として累算される（図１７は対応するアドレスを示す）。

【００９５】２つの累算器は、倍精度ステップ演算の結果をストアするためにさらに用いられる。結果の最上位部は、上位ＭＡＣのガードされる累算器ＡＧにストアされる。結果の最下位部は、下位ＭＡＣの累算器Ａにストアされる。下位ＭＡＣ累算器のガードビットは使用されない。

【００９６】各累算器は、レジスタ空間に、上位および下位累算器アドレスまたは上位および下位冗長アドレスと呼ばれる２つのアドレスを有する。（累算器ｎのためのこれらのアドレスのアセンブリ言語名はそれぞれＡｎＨおよびＡｎＬである。）どちらのアドレスが使用されるかということの効果は、レジスタが命令においてどのように用いられるかに依存し、これらの効果は以下のサブセクションにおいて詳細に述べられる。

【００９７】命令フォーマット（およびアセンブリ言語）はアドレス指定累算器の幾つかの方法を提供する。

【００９８】・レジスタ空間の要素として。各累算器は、１１１ないし１２７の範囲に、アセンブリ言語記号をＡＲｎＨおよびＡＲｎＬとする上位アドレスおよび下位アドレスを有する。

【００９９】・累算器オペランドとして。命令フォーマットは範囲０－７にある数をとり、対応するアセンブリ言語記号はＡｎ形式である。

【０１００】・別々の上位アドレスおよび下位アドレスを有する累算器オペランドとして。命令フィールドは範囲０－１５にある値をとり、アセンブリ言語フォーマットはＡｎＨまたはＡｎＬである。

【０１０１】８つのガードレジスタの各々は拡張レジスタ空間にアドレスを有する（１６０－１６７；アセンブリ言語記号はＡＧｎ形式を有する）。

【０１０２】このセクションの残りのサブセクションは、累算器およびガードレジスタの、命令としての取扱いを特定する。レジスタがソースであるかまたは宛先であるか、および演算の要素がワードであるかまたはハーフワード対であるかによって、多数の特別な例がある。

【０１０３】１．ワードソースオペランドとしての累算器

上位累算器アドレスは累算器Ａｎの上位３２ビットを小数ワードオペランドとして特定し、下位アドレスはＡｎの下位３２ビットを整数ワードオペランドとして特定する。プロセッサの現在のバージョンでは、累算器は３２ビットの長さなので、両方のアドレスとも同じ３２ビットを参照する。しかしながら、一般的なプロセッサアーキテクチャはより長い累算器を可能にする。ガードビットは、累算器（アセンブリ言語Ａｎ）を３２ビットソースオペランドとして用いる命令によって無視される。命令が、ガードされる累算器（アセンブリ言語ＡＧｎ）を

用いることを、たとえば累算レジスタのためにまたはスケーラへの入力として特定する場合には、ガードビットは４０ビットソースオペランドに含まれる。

【０１０４】バス構造は、現在、各ＭＡＣからの１つの累算器レジスタが任意の所与の命令において明示されるソースオペランドとして用いられることを可能にする。

【０１０５】累算器が乗算演算のためのソースオペランドとして選択されると、３２ビットすべてが累算器によって提示される。命令はさらに、整数／小数オプションによって、乗算アレイへの入力のための下位または上位ハーフワードを選択する。

【０１０６】２．ハーフワード対ソースオペランドとしての累算器

ハーフワード対の各要素は、累算器に、あたかもワードオペランドであるかのように保持される。ハーフワード対の２つの要素は、別個のＭＡＣにある対応する累算器にストアされる。それらのそれぞれのＭＡＣ内で累算器レジスタとしてまたはスケーラへの入力として用いられるときは、それらは４０ビットソースオペランドとして用いられる。

【０１０７】それ以外の場合には、要素はハーフワード対オペランドで２つのハーフワードとしてアセンブルされる。ハーフワード対ソースオペランドが上位累算器アドレスである場合には、各要素に対し累算器の上位ハーフワードが用いられる。下位累算器アドレスが用いられる場合には、下位ハーフワードが用いられる。アドレス指定される累算器は下位ハーフワードを与え、対応する累算器は上位ハーフワードを与える。いずれのＭＡＣもハーフワード対のいずれの要素をも供給することができる。

【０１０８】３．倍精度ソースオペランドとしての累算器

累算器は倍精度ステップ演算においてのみ精度ソースオペランドのために用いられる。アドレス指定される累算器は最下位３２ビットを与え、対応するガードされる累算器は最上位４０ビットを与える。

【０１０９】４．ソースオペランドとしてのガードレジスタ

８ビットガードレジスタ（Ｇｘ）は符号拡張整数として拡張レジスタ空間から直接アクセスすることができる。ガードレジスタがハーフワード対演算のソースオペランドである場合、アドレス指定されるガードは最下位ハーフワードオペランドとなり、対応するガードは最上位ハーフオペランドとなる。両方の例において、ガードレジスタは１６ビットに符号拡張される。

【０１１０】５．ワード宛先オペランドとしての累算器

ＭＡＣを用いるワード演算では、乗算演算の３２ビット結果は、宛先累算器にストアされ、そのガードレジスタを介して符号拡張される。累算演算の４０ビット結果は宛先ガード累算器にストアされる。

【０１１１】他の、レジスタからレジスタへの命令では、結果は、宛先累算器に移動させられ、そのガードレジスタを介して符号拡張される。

【０１１２】６．ワード対宛先オペランドとしての累算器

ワード対のデータタイプの変換を特定する累算器を目標とするロード命令では、下位メモリアドレスからのワードはアドレス指定される累算器にロードされ、より上位のメモリアドレスからのワードの最下位バイトは累算器のガードレジスタにロードされる。

【０１１３】７．ハーフワード対宛先オペランドとしての累算器

２つのＭＡＣユニットを用いるハーフワード対演算では、各ＭＡＣの結果はその累算器ファイルにストアされる。宛先累算器を含むＭＡＣは下位のハーフワード対要素を処理し、その４０ビット結果はそのガードされる累算器（ＡＧ）にストアされる。対応するＭＡＣは上位のハーフワード対要素を処理し、その４０ビット結果は対応するガードされる累算器（ＡＧＣ）にストアされる。

【０１１４】他の、レジスタからレジスタへの命令では、宛先累算器のために選択される特定の累算器アドレスが、結果をどのようにストアするかを判断する。上位アドレスが用いられる場合には、最下位ハーフワードは、選択される累算器の最上位半分にロードされ、右側へゼロ拡張され、そのガードレジスタを介して符号拡張される。最上位ハーフワードは、対応する累算器の最上位半分にロードされ、右側へゼロ拡張され、そのガードレジスタを介して符号拡張される。下位アドレスが用いられる場合には、最下位ハーフワードは、選択される累算器の最下位半分にロードされ、選択される累算器の最上位半分を介し、次いでそのガードレジスタを介して符号拡張される。最上位ハーフワードは、対応する累算器の最下位半分にロードされ、上述のように符号拡張される。

【０１１５】８．倍精度オペランドとしての累算器

倍精度乗算ステップ演算の結果の最下位３２ビットは宛先累算器にストアされ、最上位４０ビットは対応するガードされる累算器にストアされる。宛先累算器のガードビットはすべてゼロにセットされる。

【０１１６】９．宛先オペランドとしてのガードレジスタ

ガードレジスタが宛先オペランドである場合、結果の８つの最下位ビットはアドレス指定されるガードレジスタにストアされる。ガードレジスタがハーフワード対演算の宛先オペランドとしてもちいられる場合には、結果の８つの最下位ビットはアドレス指定されるガードレジスタにストアされ、上位ハーフワードの８つの最下位ビットは対応するガードレジスタにストアされる。

【０１１７】乗算アレイ

ここで図１５を参照する。各ＭＡＣのための乗算アレイ

または乗算ユニットは、２つの１６ビット入力から３２ビットの積を生ずる。符号付および符号なし入力、整数および小数入力は、任意の組合せで乗算されてもよい。整数入力の場合、ソースオペランドの最下位ハーフワードが用いられる。小数入力の場合は、最上位ハーフワードが用いられる。図１８は入力のスケーリングを示し、図１９は出力スケーリングを示す。

【０１１８】２つのワードオペランドまたは１つのワードおよび１つの即値オペランドが乗算される場合には、宛先累算器を含むＭＡＣのみが用いられる。２つのＨＰオペランドまたは１つのＨＰおよび１つの即値オペランドが乗算される場合には、両方のＭＡＣが用いられ、宛先累算器を含むＭＡＣは下位のＨＰ要素を乗算する。

【０１１９】ともに用いられる２つの乗算アレイは、図１８に従ってスケーリングされる１つの１６ビット入力と１つの３２ビット入力とから４８ビットの積を生ずる。この積は、図２０および図２１に従ってスケーリングされる。

【０１２０】乗算飽和

－１．０が累算なしで（１６ビットの符号付小数として）－１．０によって乗算される場合、結果（＋１．０）は飽和して、ガードビットへのオーバフローを防ぐ。最大の正の数は累算器（Ａ）に置かれ、ガードビットはゼロにセットされる。乗算命令が累算を含む場合には、結果は飽和せず、代わりに完全な結果が宛先ガード累算器において累算されそこに置かれる。

【０１２１】乗算スケーリング

図１８、図１９、図２０および図２１は、乗算演算のためのソースオペランドおよび結果のスケーリングを示す。表は、小数点の想定された位置および任意の符号ビットの処理を示す。

【０１２２】図１８は乗算演算のためのソースオペランドのスケーリングを示す。図１９は３２ビットの積のためのスケーリングを示す。図２０および図２１は４８ビットの積のためのスケーリングを示す。（図２０（ａ）および（ｂ）は、下位および上位ＭＡＣにおいてそれぞれ右寄せされる積のスケーリングを示し、同様に図２１（ａ）および（ｂ）は左寄せされた積のスケーリングを示す。）

累算加算器

図１５を参照すると、各ＭＡＣは、累算器に入力を加算することのできる（または累算器から入力を減算することのできる）累算加算器を含む。考えられ得る入力は、乗算アレイからの積、即値オペランド、いずれかのＭＡＣからの累算器、またはワードもしくはハーフワード対含むレジスタである。

【０１２３】累算初期化特性は、ステータスレジスタ（ＳＴ）（図示せず）のＩＭＡＣ（抑止ＭＡＣ累算）ビットによって制御される。乗算／累算演算を行なう命令が実行され、ＩＭＡＣビットが真（＝１）である場合には、宛先累算器は入力オペランドに初期化され、ＩＭＡＣビットは偽（＝０）にリセットされる（実際には、宛先累算器は、入力が累算される前に０にセットされる）。

【０１２４】同様の初期化および丸め特性は、ステータスレジスタのＩＭＡＲビットによって制御される。ＩＭＡＲビットが真である間に、累算加算器演算を行なう命令が実行されると、累算レジスタは丸め係数によって置き換えられ、宛先累算器は入力オペランドに切上げビットを加えたものに初期化され、ＩＭＡＲビットは偽にリセットされる。丸め係数は、下位ハーフワードの最上位ビットにある１を除き、すべて０である。

【０１２５】いくつかの乗算命令は、累算加算器において実行される丸めオプションを含む。丸められた結果は宛先累算器の上位ハーフワードに置かれ、ゼロは下位ハーフワードに置かれる。結果は、下位ハーフワードと上位ハーフワードとの間に小数点を有すると考えられるべきであり、結果は最も近い整数に丸められ、下位ハーフワードが１／２である場合には（つまり上位ビットが１である場合には）、結果は最も近い偶数の整数に丸められる。

【０１２６】累算加算器のオーバフローはオーバフローフラグをセットしない。飽和オプションを有する累算命令に対してオーバフロー生ずると、ガードされる累算器はオーバフローの方向に従ってそれの最も大きい正の数または最も小さい負の数にセットされる。命令が飽和を特定せず、かつ誤り例外が可能化される場合には、オーバフローは誤り例外要求を生ずる。飽和を有するオーバフローおよび飽和を有しないオーバフローのために、デバッグ論理に別個の信号が送られる。

【０１２７】図２２は、累算レジスタに加算されるワードまたは累算器オペランドを示す。図２３は、累算レジスタにあるハーフワード対に加算される（レジスタまたは累算器からの）ハーフワード対オペランドを示す。

【０１２８】図２４は、累算レジスタに加算される積を示す。図２５は、累算レジスタにあるハーフワード対に加算されるハーフワード対の積を示す。

【０１２９】図２６は、右寄せオプションを用いて累算される４８ビットの積を示す。このオプションは、整数結果が所望される１６×３２積、または３２×３２積の第１のステップに適用できる。

【０１３０】図２７は、左寄せオプションを用いて累算される４８ビット積を示す。このオプションは、小数結果が所望される１６×３２積、または３２×３２積の第２のステップに適用できる。

【０１３１】図２８－図３６は、この発明の空間ベクトルデータ経路に従って実現されるであろう演算の命令のまとめである。

【０１３２】スケーラ

図１５を参照すると、スケーラユニットは、ガードされ

る累算器の全長上で、０ないし８ビット位置の右バレルシフトを行なうことができる。最上位ガードビットは空いたビットに伝搬される。

【０１３３】ガードビットと結果の最上位ビットとがすべて一致しない場合には、スケーラ命令の間にオーバフローが生ずる。（これらのビットが一致する場合には、それは、累算器の符号ビットがガードレジスタ全体を通って伝搬し、累算器のオーバフローはガードビットには生じなかったことを意味する。）

スケーラ命令はオーバフローが生じた際に結果を飽和するオプションをサポートする。この例においては、結果は、オーバフローの方向に依って、最も大きい正の数または最も小さい負の数に１つの最下位ビットを加えてものにセットされる（最上位ガードビットは、元の数が正であったかまたは負であったかを示す。）

オーバフローが生じ、飽和が特定されなかったときに、誤り例外が可能化された場合には誤り例外が生ずる。飽和のないオーバフローおよび飽和のあるオーバフローは、別個の信号でデバッグ論理に報告される。

【０１３４】累算器を正規化するのに、レジスタへの移動がスケーリングされる累算器（ＭＡＲ）を用いてもよい。累算器Ａｎを正規化するためには：

ＭＡＲ　　Ｒｘ，ＡｎＨ，＃８；８ビットでＡＧｎをスケーリングする

ＭＥＸＰ　　Ｒｃ，Ｒｘ；指数を測定する

ＳＵＢＲＵ．Ｗ．ＳＡＴ　　Ｒｃ，Ｒｃ，＃８；正規化に必要なシフト数を計算する

ＭＡＲ　　Ｒｘ，ＡｎＨ，Ｒｃ；累算器の内容を正規化する

このシーケンスの後、Ｒｃはガードされる累算器を正規化するのに必要なシフト数を含み、Ｒｘは正規化された結果を含む。

【０１３５】この発明は図１－図３６を参照して記載されてきたが、この発明の教示は当業者によって決定されるようなさまざまな処理スキームに適用されてもよいことが理解される。

【図面の簡単な説明】

【図１】（ａ）は、従来の単一命令、多重データ（ＳＩＭＤ）コンピュータの概念的な図である。（ｂ）はＳＩＭＤコンピュータに用いられる処理素子の単純な図である。

【図２】この発明を組込むであろうプログラマブルプロセッサの一般化された図である。

【図３】（ａ）は、処理ユニットのためのＡＬＵに組込まれるであろう従来の加算器の模式図である。（ｂ）および（ｃ）は、この発明を実現するであろう加算器の模式図である。

【図４】（ａ）は、処理ユニットのためのＡＬＵに組込まれるであろう従来の論理ユニットの模式図である。（ｂ）および（ｃ）は、この発明を実現するであろう論理ユニットの模式図である。

【図５】（ａ）および（ｂ）は、この発明を実連するであろう従来のシフタの模式図である。

【図６】この発明を組込むであろうシフタの図である。

【図７】この発明を組込むであろうシフタの図である。

【図８】この発明を組込むであろうシフタの図である。

【図９】従来の乗算累算器（ＭＡＣ）の単純な図である。

【図１０】ＭＡＣがこの発明をどのように組込み得るかを示す図である。

【図１１】ＭＡＣが３２×１６モードでこの発明をどのようにして組込み得るかを示す図である。

【図１２】３２×１６モードのためのＭＡＣ内の相互接続を示す図である。

【図１３】この発明を組込む処理素子の単純な機能図である。

【図１４】この発明を組込むＡＬＵおよびシフタの単純な図である。

【図１５】デュアルＭＡＣ構成を示す図である。

【図１６】累算器レジスタファイルのレイアウトを示す図である。

【図１７】（ａ）および（ｂ）は、対応する累算器アドレスを示す図である。

【図１８】乗算演算のためのソースオペランドおよび結果のスケーリングを示す図である。

【図１９】乗算演算のためのソースオペランドおよび結果のスケーリングを示す図である。

【図２０】（ａ）および（ｂ）は、乗算演算のためのソースオペランドおよび結果のスケーリングを示す図である。

【図２１】（ａ）および（ｂ）は、乗算演算のためのソースオペランドおよび結果のスケーリングを示す図である。

【図２２】累算器レジスタに加算されるワードまたは累算器オペランドを示す図である。

【図２３】累算レジスタでハーフワード対に加算されるハーフワード対オペランドを示す図である。

【図２４】累算レジスタに加算される積を示す図である。

【図２５】累算レジスタでハーフワード対に加算されるハーフワード対の積を示す図である。

【図２６】右寄せオプションを用いて累算される４８ビットの積を示す図である。

【図２７】左寄せオプションを用いて累算される４８ビットの積を示す図である。

【図２８】この発明に従って実現されるであろう命令のまとめを示す図である。

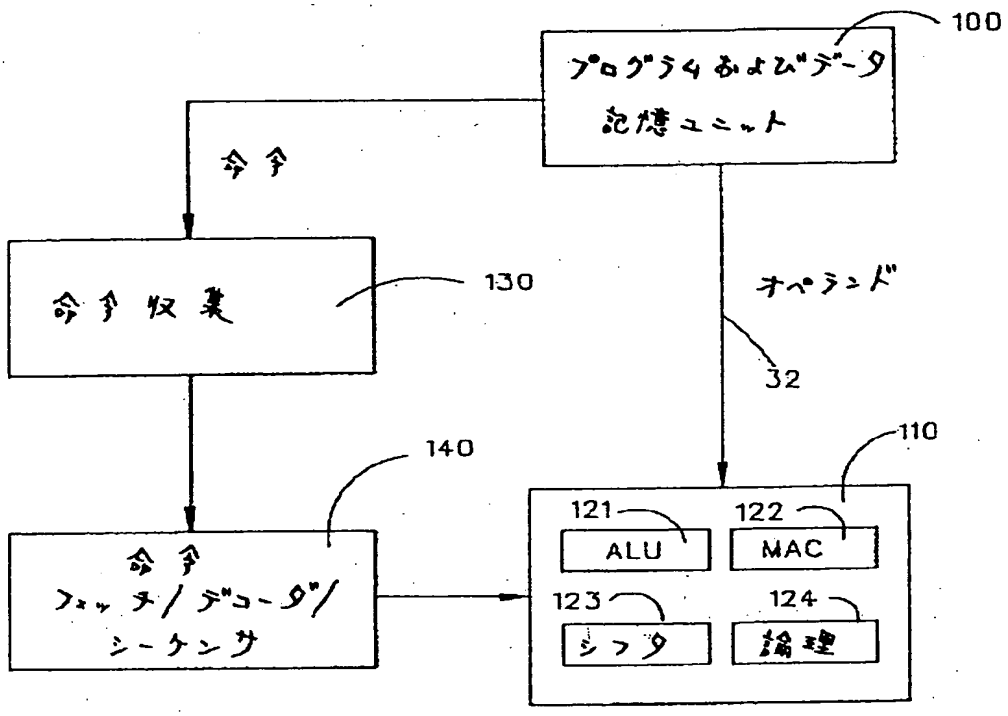
【図２９】この発明に従って実現されるであろう命令のまとめを示す図である。

【図３０】この発明に従って実現されるであろう命令の

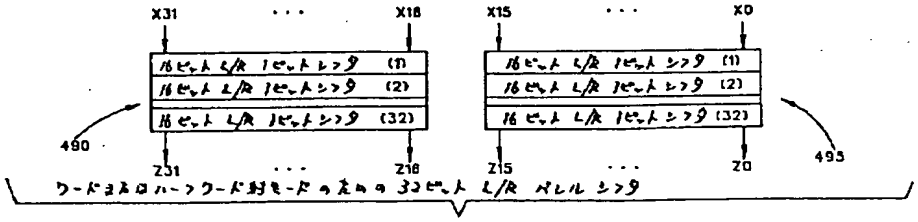
まとめを示す図である。
【図３１】この発明に従って実現されるであろう命令のまとめを示す図である。
【図３２】この発明に従って実現されるであろう命令のまとめを示す図である。
【図３３】この発明に従って実現されるであろう命令のまとめを示す図である。
【図３４】この発明に従って実現されるであろう命令のまとめを示す図である。
【図３５】この発明に従って実現されるであろう命令のまとめを示す図である。
【図３６】この発明に従って実現されるであろう命令のまとめを示す図である。

【符号の説明】
１００　プログラムおよびデータ記憶ユニット
１１０　処理ユニット
１２１　ＡＬＵ
１２２　ＭＡＣ
１２３　シフタ
１２４　論理ユニット
１３０　命令収集ユニット
１４０　命令フェッチ／デコーダ／シーケンスユニット

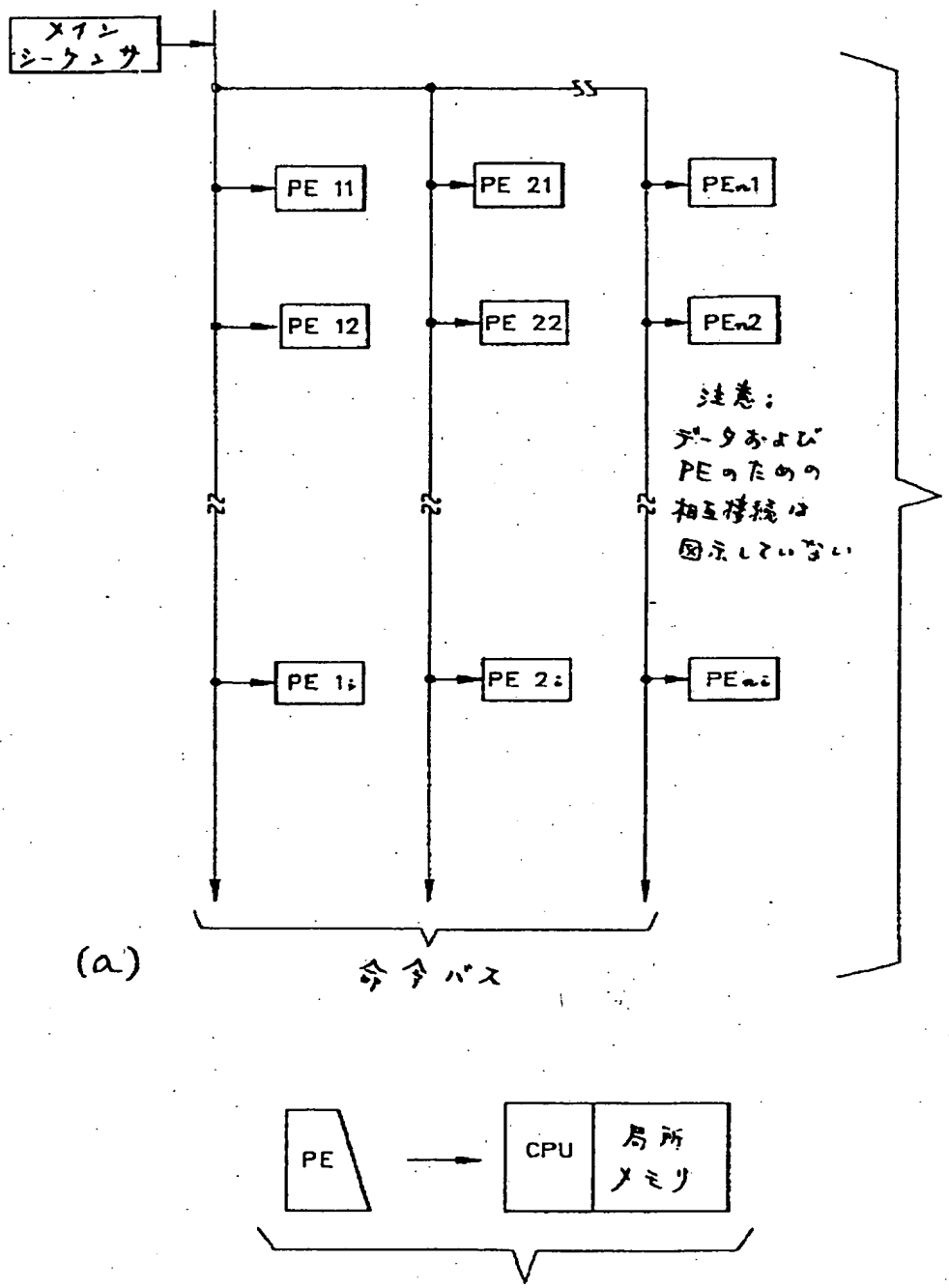
【図２】



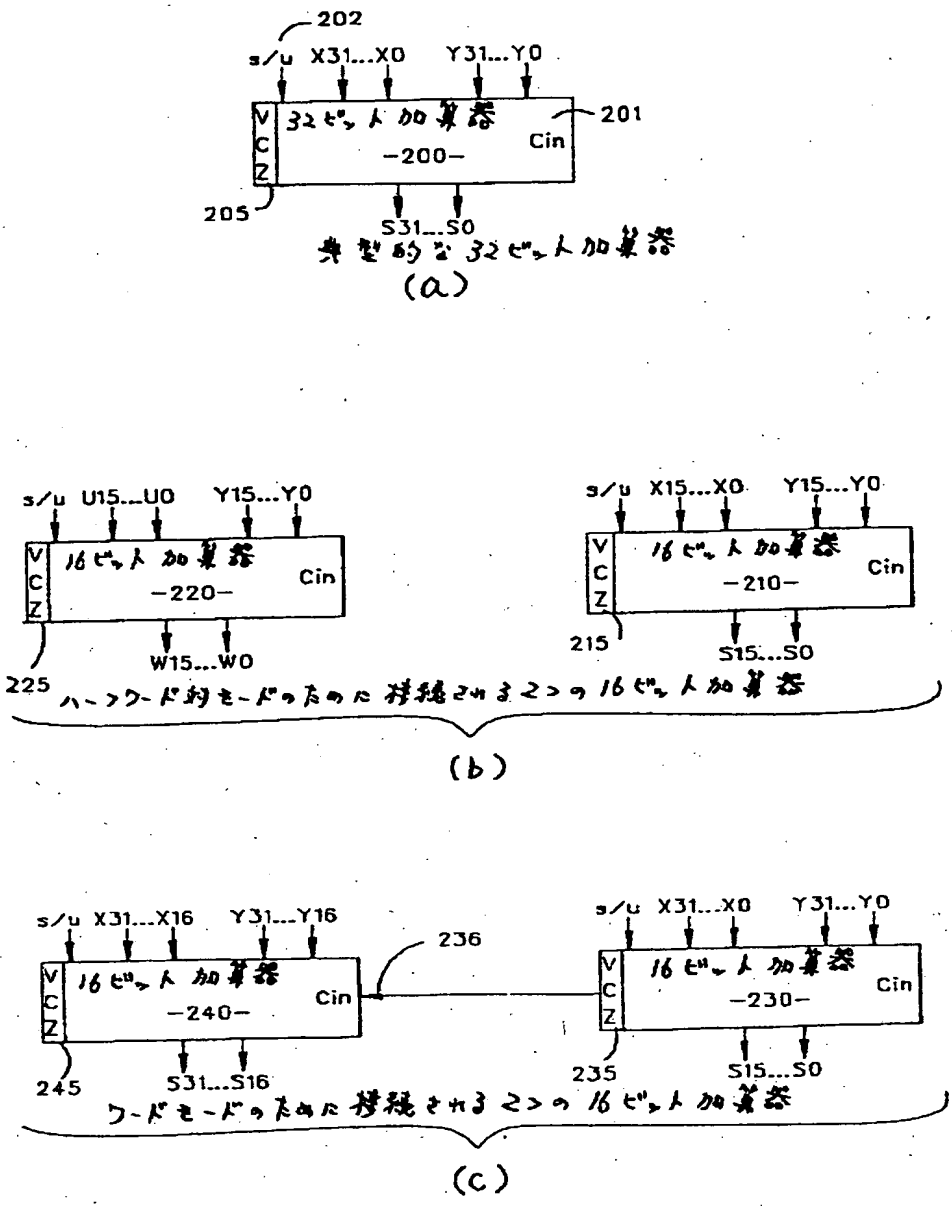

【図８】



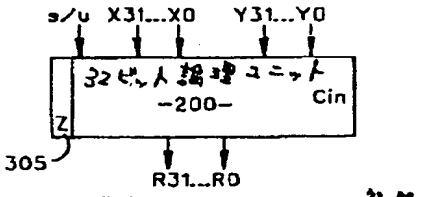
【図1】


メイン
シーケンサ
55
PE 11
PE 21
PEm1
PE 12
PE 22
PEm2
注意：
データおよび
PEのための
相互接続は
図示していない
22
22
22
PE 1i
PE 2i
PEmi
(a)
命令バス
PE
CPU
局所
メモリ
(b)

【図３】


202
s/u
X31...X0
Y31...Y0
V C Z
32ビット加算器
−200−
Cin
201
205
S31...S0
典型的な32ビット加算器
(a)
s/u
U15...U0
Y15...Y0
16ビット加算器
−220−
W15...W0
225
s/u
X15...X0
Y15...Y0
16ビット加算器
−210−
215
S15...S0
ハードワード的モードのために接続される2つの16ビット加算器
(b)
s/u
X31...X16
Y31...Y16
16ビット加算器
−240−
S31...S16
245
236
s/u
X31...X0
Y31...Y0
16ビット加算器
−230−
235
S15...S0
ワードモードのために接続される2つの16ビット加算器
(c)

【図４】

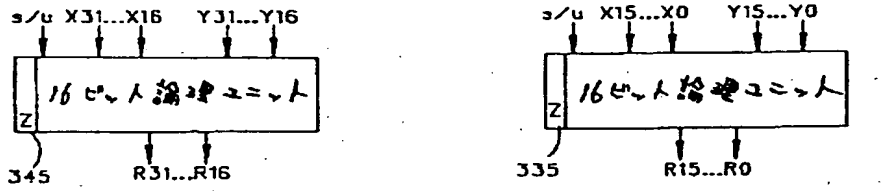



典型的な32ビット加算器

(a)


ハーフワード対モードのために接続される2つの16ビット論理ユニット

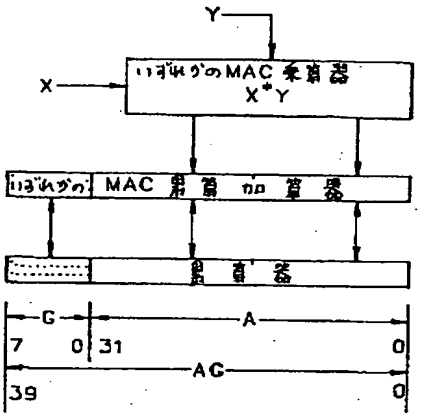
(b)




ワードモードのために接続される2つの16ビット論理ユニット

(c)

【図２４】

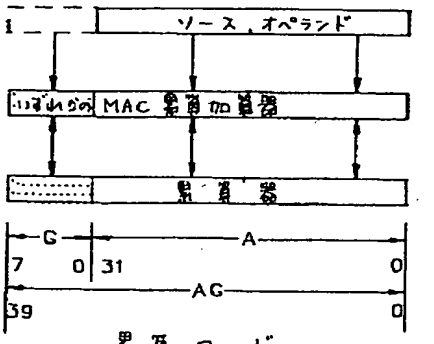



【図１６】

| | | |
|---|---|---|
| 下位 MAC 累算器 レジスタファイル | G0 | A0 |
| | G1 | A1 |
| | G2 | A2 |
| | G3 | A3 |
| 上位 MAC 累算器 レジスタファイル | G4 | A4 |
| | G5 | A5 |
| | G6 | A6 |
| | G7 | A7 |
| | <Gx(8ビット> | <——— Ax ——— (32ビット) ———> |
| | <——— AGx(40ビット) ———> | |

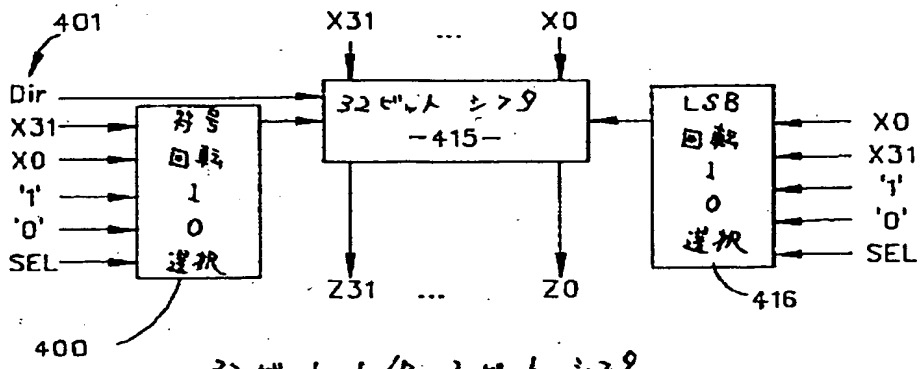
累算器レジスタ

【図２２】



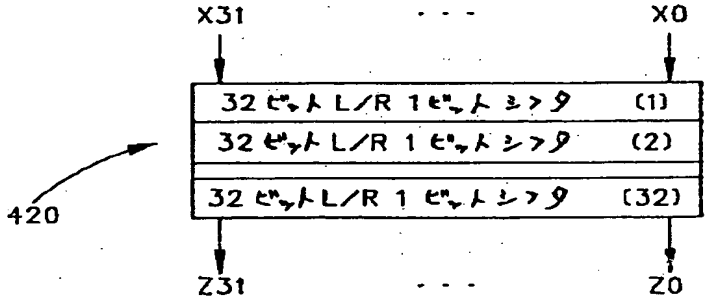

累算ワード

【図５】




32ビット L/R 1ビット シフタ

(a)



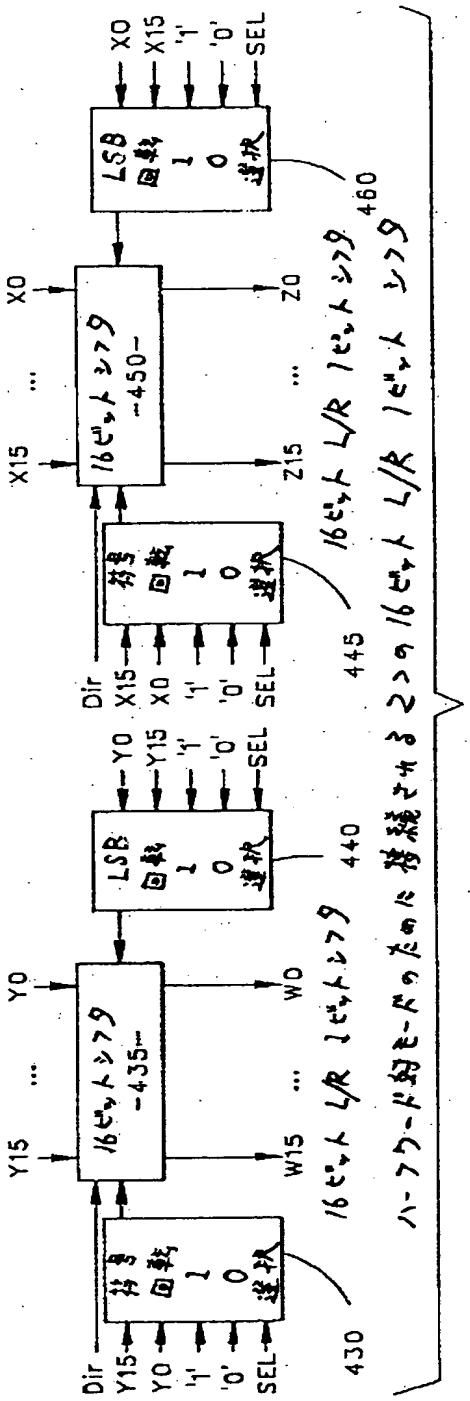

32ビット L/R バレル シフタ

(b)

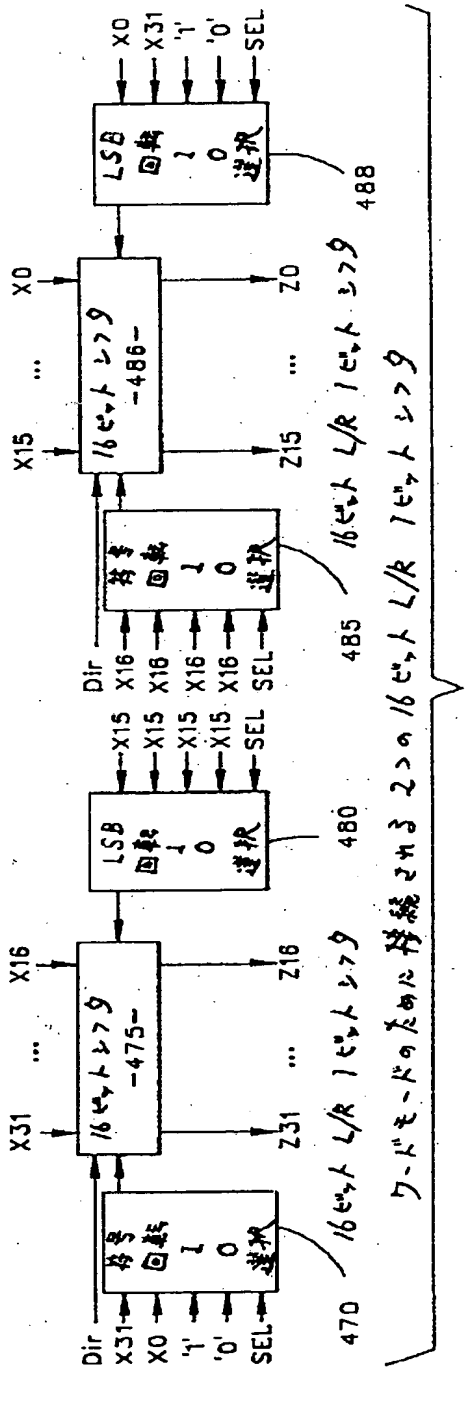
【図１８】

| 乗算器ソース オペランド フォーマット | |
|---|---|
| タイプ | 16ビット ソース オペランド フォーマット |
| SI | S u u u u u u u u u u u u u u u. |
| UI | u u u u u u u u u u u u u u u u. |
| SF | S.u u u u u u u u u u u u u u u |
| UF | .u u u u u u u u u u u u u u u |
| タイプ | 32ビット ソース オペランド フォーマット |
| SI | S u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u. |
| UI | u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u. |
| SF | S.u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u |
| UF | .u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u u |
| SI 符号付整数　UI 符号なし整数　SF 符号付小数　UF 符号なし小数　S 符号ビット　u 符号なしビット | |

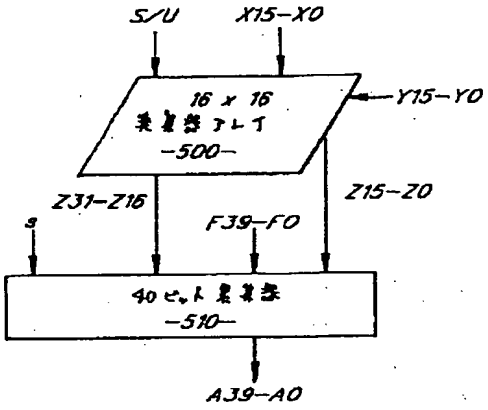
【図６】


ハーフワード幻モードの為に接続される2つの16ビットL/R 1ビットシフタ
16ビットL/R 1ビットシフタ
16ビットL/R 1ビットシフタ
-435-
-450-
LSB 回転 1 0 選択
符号 回転 1 0 選択
Dir
Y15
Y0
'1'
'0'
SEL
X15
X0
W15
W0
Z15
Z0
430
440
445
460

【図７】

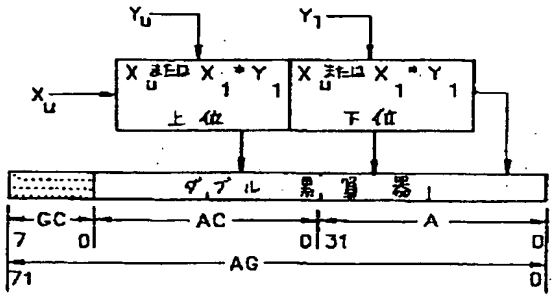

ワードモードの為に接続される2つの16ビットL/R 1ビットシフタ
16ビットL/R 1ビットシフタ
16ビットL/R 1ビットシフタ
-475-
-486-
LSB 回転 1 0 選択
符号 回転 1 0 選択
Dir
X31
X16
X15
X0
'1'
'0'
SEL
Z31
Z16
Z15
Z0
470
480
485
488

【図９】

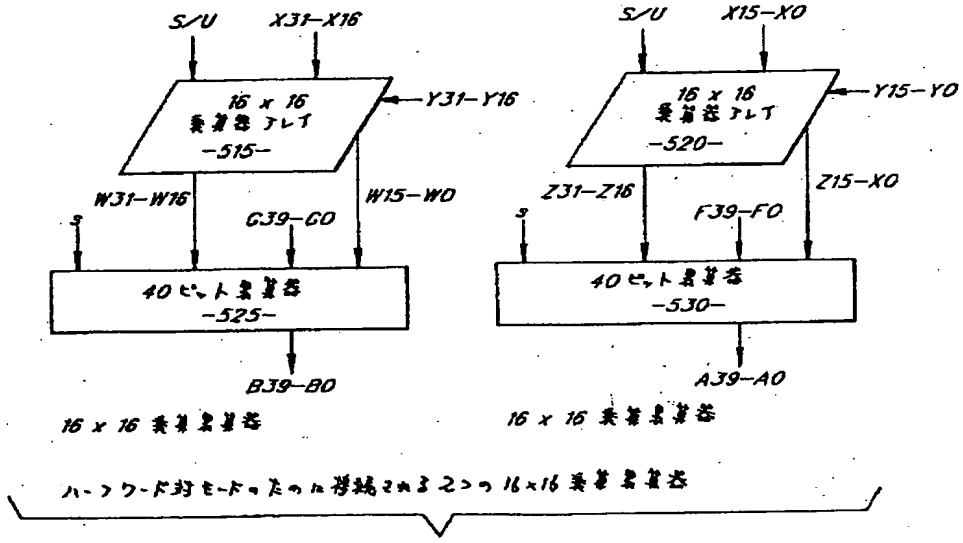



従来の16×16乗算累算器

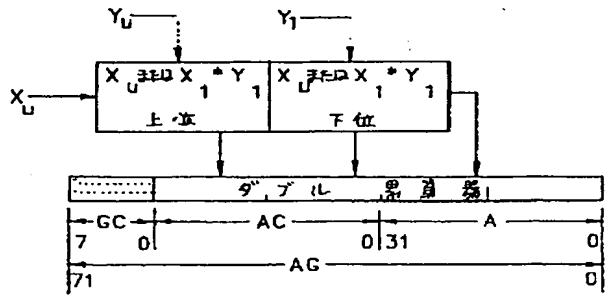
【図２６】



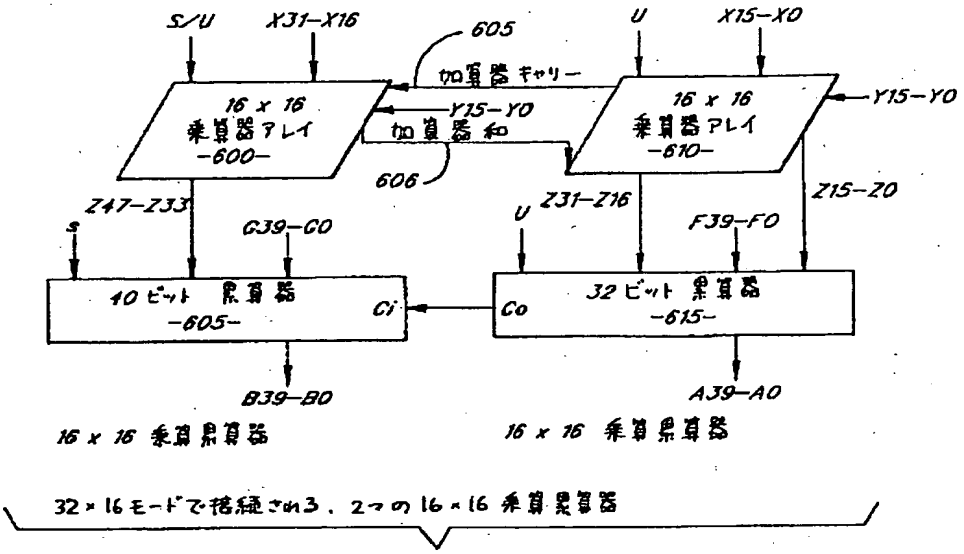

【図１０】



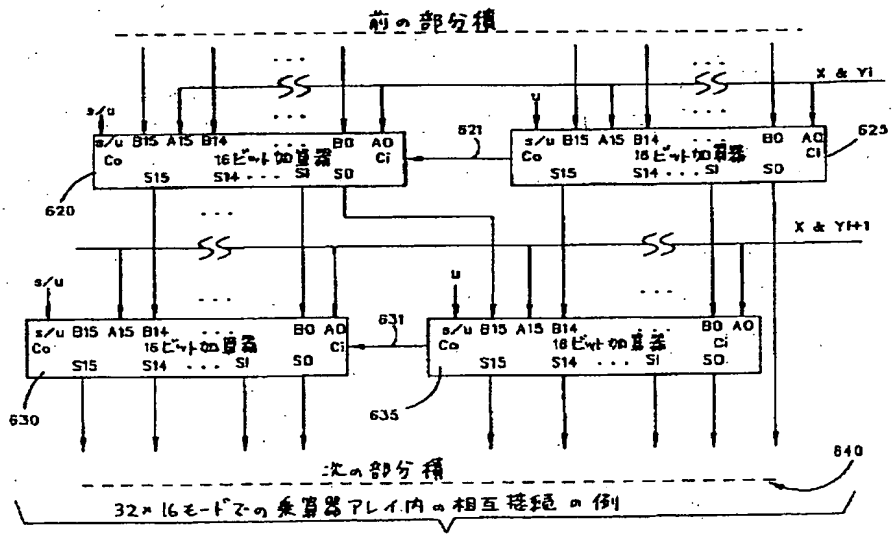

【図２７】



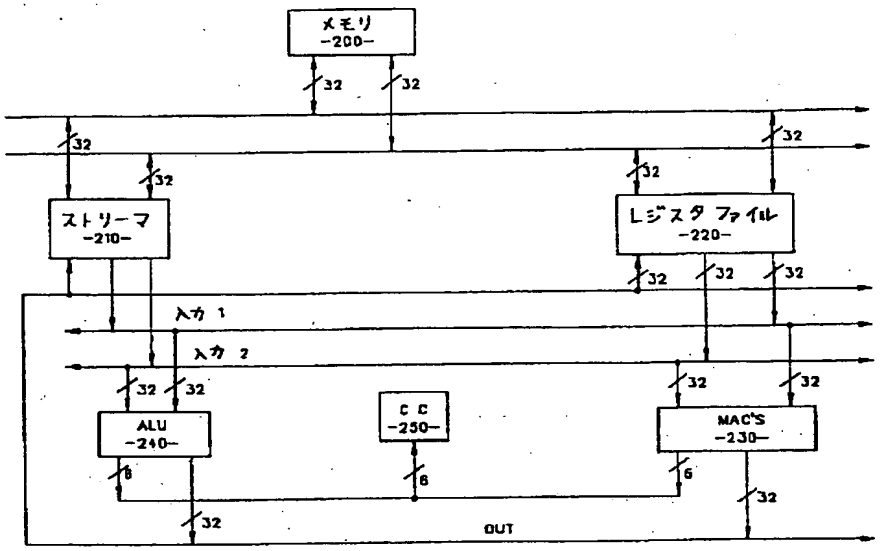
【図１１】


S/U
X31-X16
605
U
X15-X0
加算器キャリー
16 x 16
乗算器アレイ
-600-
Y15-Y0
加算器和
606
16 x 16
乗算器アレイ
-610-
Y15-Y0
Z47-Z33
G39-G0
U
Z31-Z16
F39-F0
Z15-Z0
S
40 ビット 累算器
-605-
Ci
Co
32 ビット 累算器
-615-
B39-B0
A39-A0
16 x 16 乗算累算器
16 x 16 乗算累算器
32×16モードで接続され3．2つの16×16乗算累算器

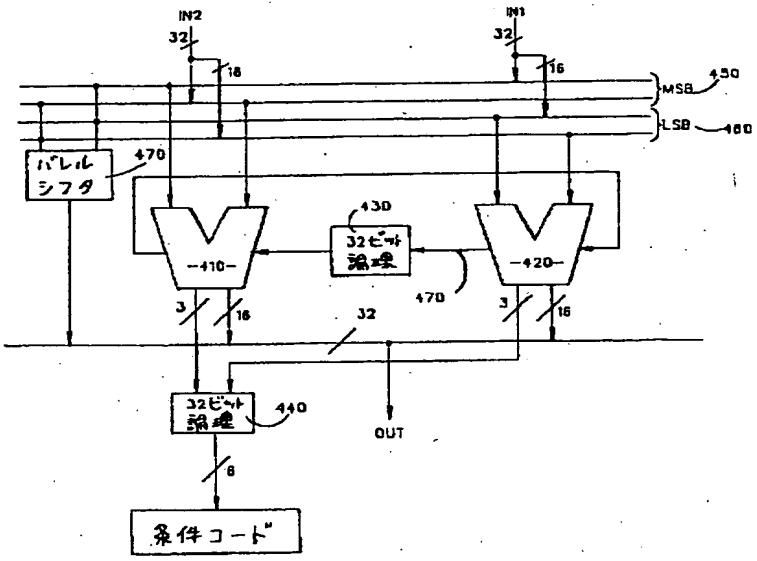
【図１２】


前の部分積
X & Yi
s/u
u
s/u B15 A15 B14 ... B0 A0
Co S15 16ビット加算器 S14 ... SI S0 Ci
621
625
620
X & Yi+1
631
630
635
次の部分積
640
32×16モードでの乗算器アレイ内の相互接続の例

【図１３】




【図１４】




【図１７】

| x | ACx | ACCx | Ax | ACx |
|---|---|---|---|---|
| 0 | AC0 | ACC4 | A0 | A4 |
| 1 | AC1 | ACC5 | A1 | A5 |
| 2 | AC2 | ACC6 | A2 | A6 |
| 3 | AC3 | ACC7 | A3 | A7 |
| 4 | AC4 | ACC0 | A4 | A0 |
| 5 | AC5 | ACC1 | A5 | A1 |
| 6 | AC6 | ACC2 | A6 | A2 |
| 7 | AC7 | ACC3 | A7 | A3 |

対応する累算器アドレス指定

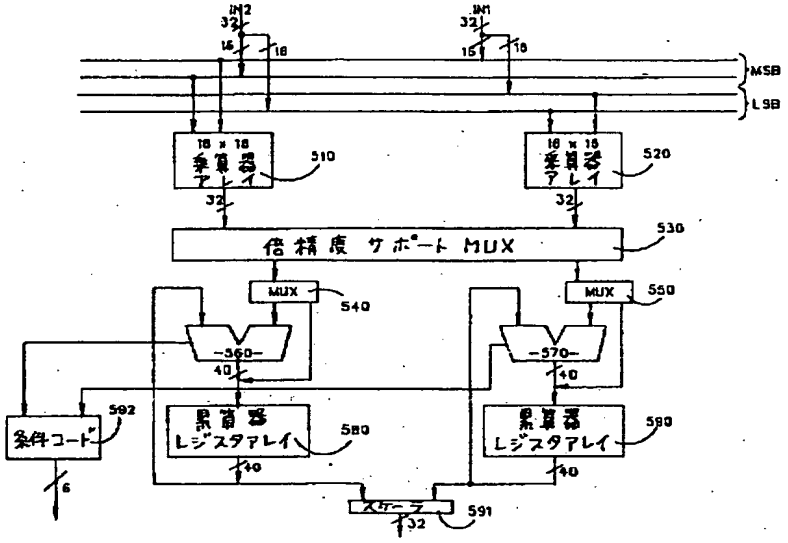
(a)

| | |
|---|---|
| SI | 符号付整数 |
| UI | 符号なし整数 |
| SF | 符号付小数 |
| UF | 符号なし小数 |
| S | 符号ビット |
| u | 符号なしビット |
| s | 符号拡張ビット |
| 0 | ゼロ拡張ビット |
| G | ガードビット |
| A | 累算ビット |

図18－図21(b)のためのキー

(b)

【図１５】




【図１９】

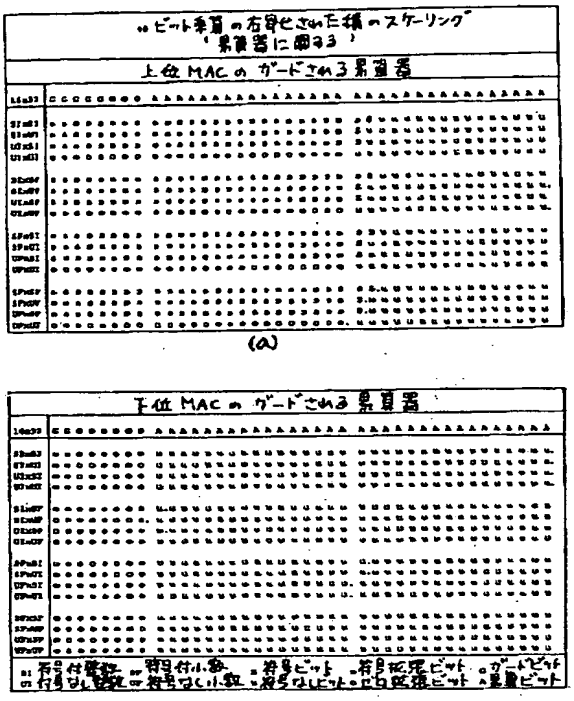
32ビット乗算積スケーリング（累算器に関する）

| 16×16 | G G G G G G G G | A A A A A A A A A A A A A A A A | A A A A A A A A A A A A A A A A |
|---|---|---|---|
| SIxSI | s s s s s s s s | s S u u u u u u u u u u u u u u | u u u u u u u u u u u u u u u u |
| SIxUI | s s s s s s s s | S u u u u u u u u u u u u u u u | u u u u u u u u u u u u u u u u |
| UIxSI | s s s s s s s s | S u u u u u u u u u u u u u u u | u u u u u u u u u u u u u u u u |
| UIxUI | o o o o o o o o | u u u u u u u u u u u u u u u u | u u u u u u u u u u u u u u u u |
| SIxSF | s s s s s s s s | s S u u u u u u u u u u u u u u | u u u u u u u u u u u u u u u u |
| SIxUF | s s s s s s s s | S u u u u u u u u u u u u u u u | u u u u u u u u u u u u u u u u |
| UIxSF | s s s s s s s s | S u u u u u u u u u u u u u u u | u u u u u u u u u u u u u u u u |
| UIxUF | o o o o o o o o | u u u u u u u u u u u u u u u u | u u u u u u u u u u u u u u u u |
| SFxSI | s s s s s s s s | s S u u u u u u u u u u u u u u | u u u u u u u u u u u u u u u u |
| SFxUI | s s s s s s s s | S u u u u u u u u u u u u u u u | u u u u u u u u u u u u u u u u |
| UFxSI | s s s s s s s s | S u u u u u u u u u u u u u u u | u u u u u u u u u u u u u u u u |
| UFxUI | o o o o o o o o | u u u u u u u u u u u u u u u u | u u u u u u u u u u u u u u u u |
| SFxSF | s s s s s s s s | S u u u u u u u u u u u u u u u | u u u u u u u u u u u u u u u u |
| SFxUF | s s s s s s s s | S u u u u u u u u u u u u u u u | u u u u u u u u u u u u u u u u |
| UFxSF | s s s s s s s s | S u u u u u u u u u u u u u u u | u u u u u u u u u u u u u u u u |
| UFxUF | o o o o o o o o | u u u u u u u u u u u u u u u u | u u u u u u u u u u u u u u u u |

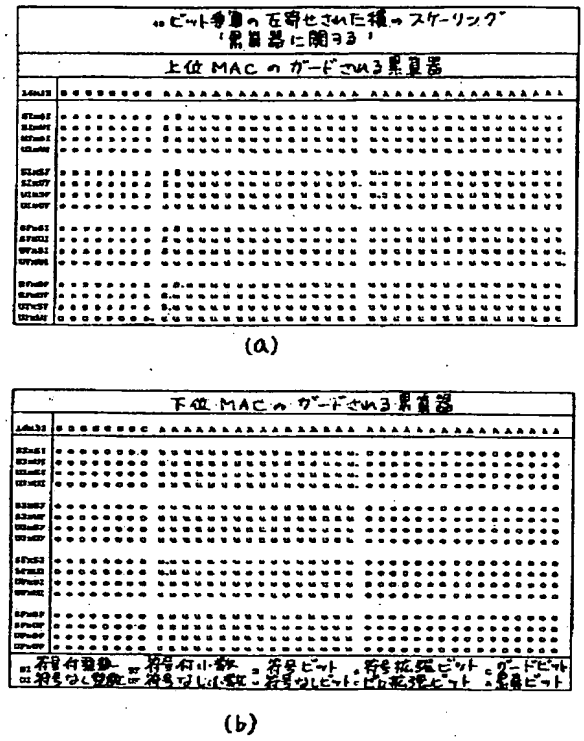
・積は、16×16、SF×SF乗算のみ、1ビット位置分左側にシフトされる。

SI 符号付整数　SF 符号付小数　s 符号ビット　s 符号拡張ビット　G ガードビット
UI 符号なし整数　UF 符号なし小数　u 符号なしビット　o ゼロ拡張ビット　A 累算ビット

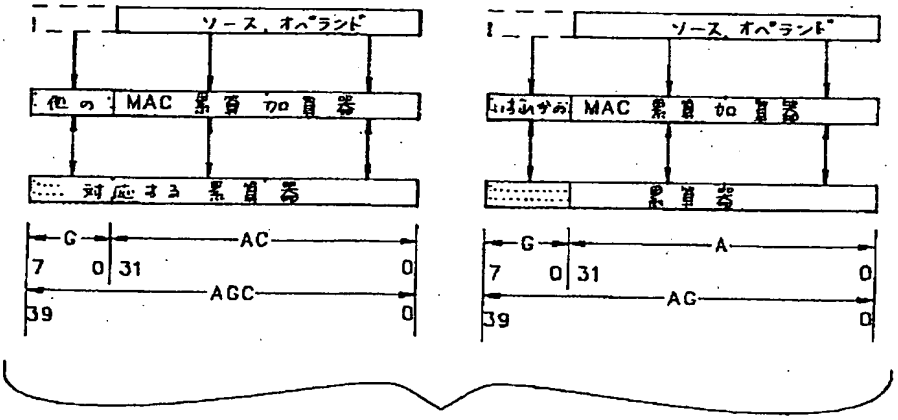
【図２０】


上位 MAC のガードされる累算器
(a)
下位 MAC のガードされる累算器
(b)

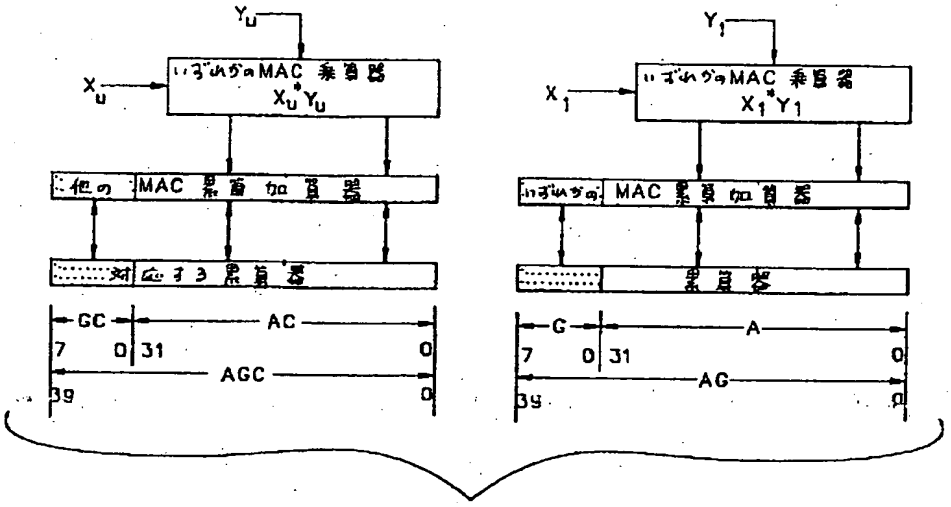
【図２１】


上位 MAC のガードされる累算器
(a)
下位 MAC のガードされる累算器
(b)

【図２３】


ソース、オペランド
他の MAC 累算加算器
対応する累算器
G
AC
7
0
31
0
AGC
39
0
ソース、オペランド
MAC 累算加算器
累算器
G
A
7
0
31
0
AG
39
0

【図２５】




【図２８】

RSP™ 命令のまとめ － 動作順序

| ニーモニック | アセンブラシンタックス | 動作 |
|---|---|---|
| L.<dsp> | L.WP dest,mem_loc | レジスタにロードする－ワード対 |
| | L[.W] dest,mem_loc | レジスタにロードする－ワード |
| | L.H[S] dest,mem_loc | レジスタにロードする－符号付ハーフワード |
| | L.HU dest,mem_loc | レジスタにロードする－符号なしハーフワード |
| | L.HP dest,mem_loc | レジスタにロードする－ハーフワード対 |
| | L.HPM dest,mem_loc | レジスタにロードする－ハーフワード対 およびマージ |
| | L.HPM dest,mem_loc | レジスタにロードする－ハーフワード対 およびマージ |
| | L.HD dest,mem_loc | レジスタにロードする－ハーフワード二重 |
| | L.B[S] dest,mem_loc | レジスタにロードする－符号付バイト |
| | L.BU dest,mem_loc | レジスタにロードする－符号なしバイト |
| | L.BF dest,mem_loc | レジスタにロードする－バイト対 |
| | L.BFM dest,mem_loc | レジスタにロードする－バイト対 およびマージ |
| | L.HP dest,mem_loc | レジスタにロードする－ハーフワード対 |
| | L.BP dest,mem_loc | レジスタにロードする－バイト対 |
| | L.BPU dest,mem_loc | レジスタにロードする－符号なしバイト対 |
| | L.BPF dest,mem_loc | レジスタにロードする－バイト対 |
| LS.<dsp> | LS.WCCM dest,src | ストリームにロードする－ワードコンテキスト、コンテキスト、およびマージ |
| | LS.WCX dest,src | ストリームにロードする－ワードコンテキスト およびインデックス |
| | LS.WPCC dest,src | ストリームにロードする－ワード対コンテキスト およびコンテキスト |
| | LS.WPCX dest,src | ストリームにロードする－ワード対コンテキスト およびインデックス |
| | LS.WPCXM dest,src | ストリームにロードする－ワード対コンテキスト、インデックス およびマージ |
| | LS.WPD dest,src | ストリームにロードする－ワード対データ |
| | LS.WPDX dest,src | ストリームにロードする－ワード対インデックス およびインデックス |
| | LS.WDX dest,src | ストリームにロードする－ワードインデックス およびインデックス |
| | LS.WDXM dest,src | ストリームにロードする－ワードインデックス、インデックス およびマージ |

【図２９】

| ニーモニック | アセンブラシンタックス | 動作 |
|---|---|---|
| S.<dsp> | S.W src,mem_loc | ワードをストアする |
| | S.H src,mem_loc | 最下位ハーフワードをストアする |
| | S.HS src,mem_loc | S.H. に対し飽和をつける |
| | S.HU src,mem_loc | S.H. に対し飽和をつける |
| | S.HP src,mem_loc | 最上位ハーフワードをストアする (小数) |
| | S.B src,mem_loc | 最上位バイトをストアする |
| | S.BU src,mem_loc | S.B. に対し飽和をつける |
| | S.BS src,mem_loc | S.B. に対し飽和をつける |
| | S.BF src,mem_loc | 最上位バイトをストアする (小数) |
| | S.HP src,mem_loc | ハーフワード対をストアする |
| | S.BP src,mem_loc | ハーフワード対の最下位バイトをストアする |
| | S.BPF src,mem_loc | ハーフワード対の最上位バイトをストアする |
| SS.<dsq> | SS.WPCX src,mem_loc | ストリームハーフワード対、コンテキスト およびインデックスをストアする |
| | SS.WPDS src,mem_loc | ストリームワード対、データ、および ステータスをストアする |
| | SS.WS | ストリームワードステータスを ストアする |
| SA.<qqp>.<sat> | SA src,mem_loc,#shift_count | スケーリングされた累算器を ストアする |
| RSE | RSE #reg_pair_count,#stack_size,#bitmask | イネーブルを予約しセットする |
| MS | MS.X dest,src | ストリームインデックスを特殊する |
| IS.<dsp> | IS.C dest,reg | ストリームコンテキストを初期化する |
| | IS.CC dest_1,dest_2,reg | ストリームコンテキストおよびコンテキストを初期化する |
| | IS.CMH dest,#imm | ストリームコンテキストを初期化してハイにマージする |
| | IS.CML dest,#imm | ストリームコンテキストを初期化してローにマージする |
| | IS.CX dest,reg | ストリームコンテキストおよびインデックスを初期化する |
| | IS.D dest | ストリームデータを初期化する |
| | IS.DD dest_1,dest_2 | ストリームデータおよびデータを初期化する |
| | IS.X dest,src | ストリームインデックスを初期化する |
| | IS.X dest,#imm | ストリームインデックスを初期化する |
| | IS.XM dest,#imm | ストリームインデックスを初期化してマージする |
| | IS.XX dest_1,dest_2,src | ストリームインデックス およびインデックスを初期化する |

【図３０】

| ニーモニック | アセンブラシンタックス | 動作 |
|---|---|---|
| LL<disp> | LL[S] dest,#imm_16<br>LLU dest,#imm_16 | レジスタに符号付即値をロードする<br>レジスタに符号なし即値をロードする |
| | LLF dest,#imm_16 | レジスタに即値小数(符号付)をロードする |
| | LLFM dest,#imm_16 | レジスタに即値小数をロードしてマージする(符号付) |
| | LLM dest,#imm_16 | レジスタに即値をロードしてマージする(符号付整数) |
| | LLD dest,#imm_16 | レジスタに倍即値をロードする(符号付整数) |
| | LLUD dest,#imm_16 | レジスタに符即値をロードする(符号付整数) |
| | LLFD dest,#imm_16 | レジスタに倍即値小数をロードする(符号付) |
| | LLMD dest,#imm_16 | レジスタに即値をロードして2倍にマージする(整数) |
| M.<qq> | M[.W]<br>M.HP | ワードと移動する<br>ハーフワード同士移動する<br>(M.H.P.に別名をつける) |
| MCC | MCC.HP dest,src<br>MCC[.W].<cc> dest,src | Cビットがクリアであれば移動する |
| MCS | MCS.HP dest,src<br>MCS[.W].<cc> dest,src | Cビットがセットされれば移動する |
| MVC | MVC.HP dest,src<br>MVC[.W].<cc> dest,src | Vビットがクリアであれば移動する |
| MVS | MVS.HP dest,src<br>MVS[.W].<cc> dest,src | Vビットがセットされれば移動する |
| MZC | MZC.HP dest,src<br>MZC[.W].<cc> dest,src | Zビットがクリアであれば移動する |
| MZS | MZS.HP dest,src<br>MZS[.W].<cc> dest,src | Zビットがセットされれば移動する |
| MZ | MZ.HP dest,src_1,src_2<br>MZ[.W].<cc1a> dest,src_1,src_2 | ゼロに等しければ移動する |
| MNZ | MNZ.HP dest,src_1,src_2<br>MNZ[.W].<cc1b> dest,src_1,src_2 | ゼロに等しくなければ移動する |
| MLT | MLT.HP dest,src_1,src_2<br>MLT[.W] dest,src_1,src_2 | より小さければ移動する |
| MGT | MGT[.W] dest,src_1,src_2 | より大きければ移動する |
| MLE | MLE[.W] dest,src_1,src_2 | より小さいかまたは等しければ移動する |
| MGE | MGE.HP dest,src_1,src_2<br>MGE[.W] dest,src_1,src_2 | より大きいかまたは等しければ移動する |

【図３１】

| ニーモニック | アセンブラシンタックス | 動作 |
|---|---|---|
| MBZ.<qq> | MBZ dest,src_1,src_2,#bit_num | ゼロ上のビットを移動する |
| MBNZ.<qq> | MBNZ dest,src_1,src_2,#bit_num | ゼロとでないビットを移動する |
| MRA.<aap> | MRA.B dest,src | レジスタから累算器へ移動する<br>－バイト |
| | MRA.H dest,src | レジスタから累算器へ移動する<br>－ハーフワード |
| | MRA.HP dest,src | レジスタから累算器へ移動する<br>－ハーフワード同 |
| | MRA[.W] dest,src | レジスタから累算器へ移動する<br>－ワード |
| MAR.<qq>.<aa> | NAR dest,src_1,src_2 | スケーリングされた累算器からレジスタへ移動する |
| PK.<qiop> | PK.HPLL dest,src_1,src_2<br>PK.HPLH dest,src_1,src_2<br>PK.HPHL dest,src_1,src_2<br>PK.HPHH dest,src_1,src_2<br>PK.BPL dest,src_1,src_2<br>PK.BP dest,src_1,src_2<br>PK.BPH dest,src_1,src_2 | 下位ハーフワード対を下位にパックする<br>下位ハーフワード対を上位にパックする<br>上位ハーフワード対を下位にパックする<br>上位ハーフワード対を上位にパックする<br>下位バイト対をパックする<br>PK.BPL.に別名をつける<br>上位バイト対をパックする |
| B.<c1> | B ofs_16 | 無条件分岐 |
| BCC.<c2>.<cc> | BCC ofs_12 | Cビットがクリアであれば分岐する |
| BCS.<c2>.<cc> | BCC ofs_12 | Cビットがセットされれば分岐する |
| BVC.<c2>.<cc> | BVC ofs_12 | Vビットがクリアであれば分岐する |
| BVS.<c2>.<cc> | BVS ofs_12 | Vビットがセットされれば分岐する |
| BZC.<c2>.<cc> | BZC ofs_12 | Zビットがクリアであれば分岐する |
| BZS.<c2>.<cc> | BZS ofs_12 | Zビットがセットされれば分岐する |
| BZ.<c2>.<cc1a> | BZ reg,ofs_12 | レジスタがゼロに等しければ分岐する |
| BNZ.<c2>.<cc1b> | BNZ reg,ofs_12 | レジスタがゼロに等しくなければ分岐する |
| BLT.<c2>.<cc2> | BLT reg,offs_12 | レジスタがゼロより小さければ分岐する |
| BGE.<c2>.<cc2> | BGE reg,ofs_12 | レジスタがゼロより大きいかまたは等しければ分岐する |
| BGT.<c2> | BGT reg,offs_12 | レジスタがゼロより大きければ分岐する |
| BLE.<c2> | BLE reg,offs_12 | レジスタがゼロより小さいかまたは等しければ分岐する |

【図３２】

| ニーモニック | アセンブラシンタックス | 動作 |
|---|---|---|
| BBZ.<a2> | BBZ reg,#bit_num,ofs_8 | ゼロに等しいビット上で分岐する |
| BBNZ.<a2> | BBNZ reg,#bit_num,ofs_8 | ゼロに等しくないビット上で分岐する |
| BBCZ.<a2> | BBCZ reg,#bit_num,ofs_8 | ゼロに等しいビットおよび補数ビット上で分岐する |
| BBCNZ.<a2> | BBCNZ reg,#bit_num,ofs_8 | ゼロに等しくないビットおよび補数ビット上で分岐する |
| BEQ.<a2> | BEQ reg_1,reg_2,ofs_8 | レジスター致すれば分岐する |
| BNE.<a2> | BNE reg_1,reg_2,ofs_8 | レジスターが等しくなければ分岐する |
| BNZD.<a2> | BNZD reg,#imm,ofs_12 | ゼロでなければ分岐し、減算する |
| BNZI.<a2> | BNZI reg,#imm,ofs_12 | ゼロでなければ分岐し、増加する |
| J.<a1><br>J.<a1>.<d1> | J addr_22<br>J (reg) | ジャンプする |
| JSB.<a1>.<d1> | JSB (reg,streamer)<br>JSB ((reg,streamer)) | ストリーマバイトモジャンプする |
| JSH.<a1>.<d1> | JSH (reg,streamer)<br>JSH ((reg,streamer)) | ストリーマハーフワードモジャンプする |
| JC.<a1>.<d1> | JC (reg_1,reg_2)<br>JC ((reg_1,reg_2)) | 条件付でジャンプする |
| JCSB.<a1>.<d1> | JCSB (streamer,reg)<br>JCSB ((streamer,reg)) | 条件付ストリーマバイトモジャンプする |
| JCSH.<a1>.<d1> | JCSH (streamer,reg)<br>JCSH ((streamer,reg)) | 条件付ストリーマハーフワードモジャンプする |
| CALL.<a1> | CALL addr_22,#reg_count<br>CALL reg,#reg_count | サブルーチンモードが出る |
| TRAP.<a1> | TRAP addr_22<br>TRAP (reg) | トラップする |
| TRAPC.<a2> | TRAPC addr_22<br>TRAPC (reg) | 条件付トラップ |

【図３３】

| ニーモニック | アセンブラシンタックス | 動作 |
|---|---|---|
| RETE.<a1> | RETE | 例外からもどる |
| RETI.<a1> | RETI | 割込からもどる<br>（RETEも同じ動きをする） |
| RETT.<a1> | RETT | トラップからもどる<br>（RETEも同じ動きをする） |
| RETQ | RETQ | 即時割込／例外からもどる |
| NOP.<cnp> | NOP[.0]<br>NOP.1 | 動かない ― 0 ビット<br>動かない ― 1 ビット |
| LOOP.<ae>.<cp> | LOOP ofs_8,reg<br>LOOP ofs_8,#loop_count | 高速ループ |
| WAIT | WAIT | 待期 |
| HALT | HALT | 停止 |
| BREAK | BREAK | ブレイク点 |
| ABS.<pqp>.<cnr> | ABS dest,src | 絶対値 |
| NEG.<pqp>.<cnp> | NEG dest,src | 負数（2の補数） |
| NOT.<pqp> | NOT dest,src | 1の補数 |
| PARE.<pqp> | PARE dest,src | 偶数パリティ，偶数 |
| PARO.<pqp> | PARD dest,src | 偶数パリティ，奇数 |
| REV.<pqp> | REV dest,src | ビット逆順 |
| ADD[S].<pqp>.<cnr> | ADD dest,src_1,src_2<br>ADD dest,src_1,#imm | 加算する（符号付） |
| ADDU.<pqp>.<cnr> | ADD dest,src_1,src_2<br>ADD dest,src_1,#imm | 符号なしで加算する |
| ADDC[S].<pqp>.<cnr> | ADDC dest,src_1,src_2 | 桁上げを持って加算する |
| ADDCU.<pqp>.<cnr> | ADDCU dest,src_1,src_2 | 符号なし桁上げを持って加算する |
| SUB[S].<pqp>.<cnr> | SUB[S] dest,src_1,src_2<br>SUB[S] dest,src_1,#imm | 減算する（符号付） |
| SUBU.<pqp>.<cnr> | SUBU dest,src_1,src_2<br>SUBU dest,src_1,#imm | 符号なしで減算する |
| SUBC[S].<pqp>.<cnr> | SUBC[S] dest,src_1,src_2 | 桁上げを持って減算する（符号付） |
| SUBCU.<pqp>.<cnr> | SUBCU dest,src_1,src_2 | 符号なし桁上げを持って減算する |
| SUBR[S].<pqp>.<cnr> | SUBR[S] dest,src_1,#imm | 減算と逆の符（符号付） |

【図３４】

| ニーモニック | アセンブラシンタックス | 動作 |
|---|---|---|
| SUBRU.<qqp>.<cc> | SUBRU dest,src_1,#imm | 符号なし減算を行う |
| ASC.<qqp> | ASC dest, src_1,src_2,#bit_num | 条件付き加算/減算 |
| MIN[S] | MIN dest,src_1,src_2 | 最小値（符号付） |
| MINU | MINU dest,src_1,src_2 | 符号なし最小値 |
| MAX[S].<qqp> | MAX dest,src_1,src_2 | 最大値（符号付） |
| MAXU.<qqp> | MAXU dest,src_1,src_2 | 符号なし最大値 |
| TEQ.<qqp> | TEQ dest,src_1,src_2<br>TEQ dest,src_1,#imm | 0と等しいレジスタをテストする |
| TNE.<qqp> | TNE[S] dest,src_1,src_2<br>TNE[S] dest,src_1,#imm | 0と等しくないレジスタをテストする |
| TLT[S].<qqp> | TLT[S] dest,src_1,src_2<br>TLT[S] dest,src_1,#imm | 0未満のレジスタをテストする（符号付） |
| TGT[S].<qqp> | TGT[S] dest,src_1,src_2<br>TGT[S] dest,src_1,#imm | 0より大きいレジスタをテストする（符号付） |
| TLE[S].<qqp> | TLE[S] dest,src_1,src_2<br>TLE[S] dest,src_1,#imm | 0以下のレジスタをテストする（符号付） |
| TGE[S].<qqp> | TGE[S] dest,src_1,src_2<br>TGE[S] dest,src_1,imm | 0以上のレジスタをテストする（符号付） |
| TLTU.<qqp> | TLTU dest,src_1,src_2<br>TLTU dest,src_1,#imm | 符号なし、0未満のレジスタをテストする |
| TGTU.<qqp> | TGTU dest,src_1,src_2<br>TGTU dest,src_1,#imm | 符号なし、0より大きいレジスタをテストする |
| TLEU.<qqp> | TLEU dest,src_1,src_2<br>TLEU dest,src_1,#imm | 符号なし、0以下のレジスタをテストする |
| TGEU.<qqp> | TGEU dest,src_1,src_2<br>TGEU dest,src_1,#imm | 符号なし、0以上のレジスタをテストする |
| TAND.<qqp> | TAND dest,src_1,src_2<br>TAND dest,src_1,#imm | ビット単位 AND の結果をテストする |
| TOR.<qqp> | TOR dest,src_1,src_2<br>TOR dest,src_1,#imm | ビット単位 OR の結果をテストする |
| SBIT.<qqp> | SBIT dest,src_1,src_2 | ビットをセットする |
| CBIT.<qqp> | CBIT dest,src_1,src_2 | ビットをクリアする |
| IBIT.<qqp> | IBIT dest,src_1,src_2 | ビットを反転する |
| TBZ.<qqp> | TBZ dest,src_1,src_2<br>TBZ dest,src_1,#imm | ビットのをテストする |

【図３５】

| ニーモニック | アセンブラシンタックス | 動作 |
|---|---|---|
| TBNZ.<qqp> | TBNZ dest,src_1,src_2<br>TBZ dest,src_1,#imm | ビットがビットをテストする |
| AND.<qqp> | AND dest,src_1,src_2<br>AND dest,src_1,#imm | ビット単位 AND |
| ANDN.<qqp> | ANDN dest,src_1,src_2<br>ANDN dest,src_1,src_2 | ビット単位 AND - NOT |
| OR.<qqp> | OR dest,src_1,src_2<br>OR dest,src_1,#imm | ビット単位 OR |
| XOR.<qqp> | XOR dest,src_1,src_2 | ビット単位 XOR |
| XORC.<qqp> | XORC dest,src_1,src_2 | 条件付ビット単位XOR |
| SHR.<qqp> | SHR dest,src_1,src_2<br>SHR dest,src_1#imm | 論理右シフトする |
| SHL.<qqp> | SHL dest,src_1,src_2<br>SHL dest,src_1,#imm | 論理左シフトする |
| SHRA.<qqp> | SHRA dest,src_1,src_2<br>SHRA dest,src_1,#imm | 算術右シフトする |
| SHRC.<qqp> | SHRC dest,src_1,src_2<br>SHRC dest,src_1,#imm | 物ムダを用いて右シフトする |
| ROR.<qqp> | ROR dest,src_1,src_2<br>ROR dest,src_1,#imm | 右へ回転する |
| ROL.<qqp> | ROL dest,src_1,src_2<br>ROL dest,src_1,#imm | 左へ回転する |
| INS | INS dest,src,#shift_count,#bit_count | 挿入する |
| EXT | EXT dest,src,#shift_count,#bit_count | 抽出する |
| CNT | CNT.LSO dest,src | 下位のビットまをカウントする |
| | CNT.LSZ dest,src | 下位のビットのまをカウントする |
| | CNT.LSR dest,src | ビットの最下位までまをカウントする |
| | CNT.MSO dest,src | 上位のビットまをカウントする |
| | CNT.MSZ dest,src | 上位のビットのをカウントする |
| | CNT.MSR dest,src | ビットの最上位ランまをカウントする |

【図３６】

| ニーモニック | アセンブラシンタックス | 動作 |
|---|---|---|
| CEXP | CEXP dest,src_1,src_2 | 指数を比較する |
| MEXP | MEXP dest,src | 指数を測定する |
| SYM | SYM dest,src,#table_type,#shift_count | 対称テーブルアドレス |
| ACC | ACC.<sz> dest, src_1,src_2<br>ACC.<qqp>.<sz> dest,src_1<br>ACC.<qqp>.<sz> dest,#imm | 累算する |
| ACCN | ACCN.<sz> dest,src_1,src_2<br>ACCN.<qqp>.<sz> dest,src_1<br>ACCN.<qqp>.<sz> dest,#imm | 負に累算する |
| MUL.<qqp>.<sz> | MUL dest,src_1, src_2<br>MUL dest,src_1,#imm | 乗算する |
| MAC.<qqp>.<sz> | MAC dest_1,src_1,src_2,src_3 | 乗算および累算する |
| MACN.<qqp>.<sz> | MACN dest_1,src_1,src_2,dest_3 | 負の乗算および累算する |
| UMUL.<qqp>.<sq> | UMUL dest,src_1,src_2 | ユニバーサルハーフワード符号乗算する |
| UMAC.<qqp>.<sq> | UMAC dest_1,dest_2,src_1,src_2 | ユニバーサルハーフワード符号乗算および累算する |
| DMUL.<jq> | DMUL dest,src_1,src_2 | 2倍乗算ステップ |
| DMULN.<jq> | DMULN dest,src_1,src_2 | 負の2倍乗算ステップ |
| DMAC.<jq> | DMAC dest,src_1,src_2,src_3 | 2倍乗算および累算ステップ |
| DMACN.<jq> | DMACN dest,src_1,src_2,src_3 | 負の2倍乗算および累算ステップ |

フロントページの続き

(72)発明者　ケニス・イー・ギャレイ
アメリカ合衆国、92714　カリフォルニア州、アーバイン、フレンズ・コート、17531

(72)発明者　ジョージ・エイ・ワトソン
アメリカ合衆国、92635　カリフォルニア州、フラートン、ツリービュー・プレイス、2952

(72)発明者　ジョン・アール
アメリカ合衆国、92680　カリフォルニア州、タスティン、ウィリアムズ・ストリート、15512－ビー